



Spike2 version 8 for Windows

Copyright © Cambridge Electronic Design 1995-2014

Spike2 version 8 for Windows

The Spike2 online help as a manual

by Cambridge Electronic Design Limited

Spike2 version 8 for Windows

Copyright © Cambridge Electronic Design 1995-2014

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the prior written permission of Cambridge Electronic Design (CED) Limited.

Permission is granted to make a backup copy for security purposes. Permission is granted to print copies of this documentation for use by the licensee. Permission is granted to use attributed extracts from this documentation for educational purposes. Commercial copying, hiring or lending is prohibited.

While every precaution has been taken in the preparation of this document, CED assumes no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that accompany it. In no event shall CED be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document or software.

Printed: June 2014 in Cambridge, England

Revision History

Version 8.00	Dec 2013
Version 8.01	Jan 2014
Version 8.02	Jun 2014

Published by:

Cambridge Electronic Design Limited
Science Park
Milton Road
Cambridge
CB4 0FE
UK

Telephone: Cambridge (01223) 420186
International: +44 1223 420186
Fax: Cambridge (01223) 420488
International Fax: +44 1223 420488
Email: info@ced.co.uk
Home page: www.ced.co.uk

Acknowledgements

Curve fitting functions are based on routines in Numerical Recipes: The Art of Scientific Computing, published by Cambridge University Press and are used by permission.

The XML library used to save and restore resources is written by Michael Chourdakis (www.turboirc.com).

Trademarks and Tradenames used in this document are acknowledged to be the Trademarks and Tradenames of their respective Companies and Corporations.

Table of Contents

Spike2 version 8.....	1-2
CED Software Licences	1-4
Installation	1-4
Updating and removing Spike2	1-5
Versions of Spike2	1-5
Getting started with Spike2.....	2-8
Demonstration script	2-8
Opening a file to view	2-8
Data channels	2-9
Zoom buttons	2-9
X axis short cut keys	2-9
Cursor buttons	2-10
Cursor style and pointers	2-10
Active cursor modes	2-10
Automatic measurements to XY view	2-11
Zoom in on an area	2-11
Zoom a channel	2-11
Using x and y axes to scroll and zoom	2-12
X Range dialog	2-12
Y Range dialog	2-13
Channel draw mode dialog	2-13
Show and Hide channels	2-13
Channel order	2-14
Channel spacing	2-14
Channel overdraw	2-14
Cursor values	2-15
Cursor regions	2-15
Result views	2-16
Make an interval histogram	2-16
Process dialog	2-16
Repeating and extending a process	2-16
Result view drawing modes	2-17
Result view cursors	2-17
Other sources of information	2-17
General information.....	3-20
Spike2 clock tick	3-20
Data file versions	3-20

- Data channel types 3-21
- Channel lists 3-23
- Dialog expressions 3-23
- Copy current file path to clipboard 3-25
- Data view keyboard shortcuts 3-25
- Text view keyboard shortcuts 3-26
- Text view features 3-28
- The Spike2 command line 3-29
- Shell extensions 3-30
- 64-bit operating systems 3-30
- Mouse buttons 3-31
- Program files installation 3-31

Sampling data.....4-34

- Sampling configuration 4-34
- Data buffering 4-62
- Opening a new document 4-63
- Process dialog for a new file 4-64
- Sample control toolbar 4-64
- High sampling rates 4-65
- The Sample Status bar 4-66
- Saving configurations 4-66
- Sequence of operations to set the configuration 4-67

Output sequencer.....5-70

- Overview 5-70
- The graphical editor 5-73
- The text editor 5-83
- Instructions 5-89

File menu.....6-122

- New File 6-122
- Other types used by Spike2 6-122
- Open 6-123
- Read only files 6-123
- Import 6-124
- Global Resources 6-129
- Resource Files 6-130
- Close, Close and Link 6-130
- Revert To Saved 6-131
- Save and Save As 6-131
- Export As 6-132
- Load and Save Configuration 6-140
- Exit 6-141

Send Mail	6-142
Printing	6-143

Edit menu.....7-148

Undo and Redo	7-148
Cut	7-148
Copy	7-148
Paste	7-153
Delete	7-153
Clear	7-153
Select All	7-153
Find, Find Again, Find Last	7-154
Replace	7-155
Edit toolbar	7-155
Auto Format	7-156
Toggle Comments	7-156
Auto Complete	7-157
Preferences	7-158

View menu.....8-172

Toolbar and Status bar	8-172
Enlarge View Reduce View	8-172
X Axis Range	8-173
Y Axis Range	8-174
Standard Display	8-176
Show/Hide Channel	8-176
Vertical Markers	8-177
Pen Width	8-179
Info	8-179
File Information	8-179
Channel Information (Time view)	8-179
Channel Information (Result view)	8-180
Channel Image	8-181
Options	8-182
Trigger/Overdraw	8-182
Channel Draw Mode	8-188
Multimedia files	8-197
Spike Monitor	8-198
Font	8-198
Use Colour and Use Black And White	8-199
Change Colours	8-199
Sonogram Colours	8-202
Folding	8-203
Show Gutter	8-203

Show Line Numbers	8-204
ReRun	8-204
Annotate	8-204

Analysis menu.....9-206

New Result View	9-206
Measurements to XY views	9-217
Measurements to a data channel	9-222
Fit Data	9-223
Memory buffer	9-228
Virtual Channels	9-232
Duplicate Channels	9-244
Save channel	9-245
Delete channel	9-245
Calibrate	9-245
Channel process	9-247
Marker Filter	9-251
Set Marker Codes	9-252
New WaveMark	9-253
New Stereotrode, Tetrode	9-253
Edit WaveMark	9-253
Digital filters	9-253

Window menu.....10-256

Duplicate window	10-256
Hide	10-256
Show	10-256
Tile Horizontally	10-256
Tile Vertically	10-256
Cascade	10-256
Arrange Icons	10-257
Close All, Close All and Link	10-257
Windows	10-257

Cursor menu.....11-260

Vertical cursors	11-260
Horizontal cursors	11-268
Cursor context commands	11-269

Sample menu.....12-272

Sampling configuration	12-272
Clear configuration	12-272
Sample bar	12-272
Conditioner Settings	12-273
Sampling controls	12-273

Sampling Notes	12-273
Talker list	12-273
Sequencer controls	12-274
Create a TextMark	12-274
Change Output Sequence	12-275
Offline waveform output	12-275
Script menu.....	13-280
Compile Script	13-280
Run Script	13-280
Evaluate	13-280
Turn Recording On Off	13-280
Debug bar	13-281
Script bar	13-281
Help menu.....	14-284
Using help	14-284
Tip of the Day	14-284
View Web site	14-284
Getting started	14-284
Other sources of help	14-284
About Spike2	14-285
Script language.....	15-288
Introduction to scripting	15-288
Script window and debugging	15-292
Script language syntax	15-296
Script functions by topic	15-316
Alphabetical script function index	15-329
Curve fitting	15-628
XY Views	15-631
Spike sorting.....	16-638
Introduction	16-638
Spike shape sorting dialogs	16-640
On-line template setup	16-642
Selecting the area for a template	16-643
Horizontal cursors	16-644
The template area	16-645
Edit template code	16-646
Merging templates	16-646
Manual template creation	16-646
Toolbar controls	16-646
Multiple traces	16-648
Template formation	16-649

Template settings dialog	16-650
Off-line template formation	16-652
Print templates and Copy	16-653
Off-line template editing	16-655
Collision Analysis Mode	16-656
Analyse menu commands	16-658
On line template monitoring	16-659
Load Save templates	16-660
Spike Monitor	16-662
Getting started with spike shapes and templates	16-664
Creating on line templates with cluster analysis	16-672

Clustering.....17-676

Introduction	17-676
Principal Component Analysis	17-677
Cluster on measurements	17-679
Cluster on template correlations	17-682
Cluster on template errors	17-683
The Clustering dialog	17-683
Menu commands	17-689
Getting started with clustering	17-701
K Means algorithm	17-703
Normal Mixtures algorithm	17-704
Mahalanobis distance	17-706

Digital filtering.....18-708

FIR and IIR filters	18-708
Digital filter dialog	18-709
Filter bank	18-710
FIR filter details	18-711
IIR filter details	18-714
FIR filters and scripts	18-717

Programmable signal conditioners.....19-728

What a signal conditioner does	19-728
Serial ports	19-728
Control panel	19-728
Setting the channel gain and offset	19-730
Conditioner connections	19-731

Test and utility programs.....20-734

The S64Fix data recovery utility	20-734
The SonFix data recovery utility	20-737
Try1401 test program	20-741

Multimedia recording.....	21-746
The s2video application and file names	21-746
System requirements	21-747
Getting started	21-747
Video Device Properties	21-748
Audio Device Properties	21-748
Video Capture	21-748
Set Slow Frame Rate	21-749
Use Slow Frame Rate	21-749
Configuration	21-749
View menu	21-753
File menu	21-753
Recording data	21-754
The avicom application	21-754
Technical support.....	22-758
How to contact CED	22-758
Recent Spike2 revision history	22-759
Frequently asked questions	22-764
Index.....	Index-1

1: Spike2 version 8

Spike2 version 8

With Spike2 version 8 and a Power1401, Power1401 mk II, Power1401-3, Micro1401 mk II or Micro1401-3, you can capture and analyse waveform, event and marker data and output precisely timed pulses and voltages using the familiar and easy to use Windows environment. If you have a 1401*plus* or the original micro1401 you can capture data with Spike2 version 7, which is also on the installation medium.

You can arrange the windows to display the data within them to best advantage and cut and paste the results to other applications. Alternatively, you can obtain printer hard copy directly from the application. When you close a data file, Spike2 saves the screen format and channel display settings. When you open a file, Spike2 restores the configuration, so it is easy to resume work where you stopped in a previous session.

You can analyse sections of data by reading off values at and between cursors, or by applying the built-in functions, for example waveform averaging, digital filtering, spike detection, histogram formation and power spectra. More ambitious users can automate both data capture and analysis with scripts and the output sequencer.

New features in version 8

We have tried very hard to keep version 8 of Spike2 compatible with version 7. It reads data files from all previous versions. Resource files are mostly compatible; some resource formats have changed to support new features. Scripts that ran with version 7 should work unchanged with version 8; there can be differences due to integers now being 64 bits and because of the new filing system. New features in version 8 include:

- New data file format that allows files of effectively unlimited size (much bigger than any available disk)
- Sampling times are no longer limited to 2 billion clock ticks (time stamps are 64-bits, they were 32-bits)
- You can install 64-bit code on 64-bit systems, which makes more memory available and runs faster
- You can sample to the new 64-bit file format or use the old 32-bit format for backwards compatibility
- Sonogram displays are improved and can now have a key
- Extended Talker support with interactive script support
- Script editor improvements for call tips and go to definition (even in included files)
- Script language integers are now 64-bits

There are many other improvements and more are planned. You can find a full list of new features, bug fixes and changes in the Revision History. Licensed users of Spike2 version 8 can download updates of version 8 from our Web site www.ced.co.uk as they become available.

Upgrading from earlier versions

Sampling configurations generated by Spike2 version 7 or earlier will write 32-bit *.smr files for backwards compatibility. You can change to the new format for sampling in the Sampling Configuration dialog Resolution tab. Although we allow sampling to old format files, the program is optimised for the new 64-bit *.smrx file format. Sampling to the 32-bit format is slower as we must convert data from 64-bit to 32-bit; it may be slower than using Spike2 version 7 to sample. All our efforts to improve sampling in version 8 are concentrated on the 64-bit format. Unless you have pressing reasons for sticking with the 32-bit format we encourage you to migrate to the 64-bit format.

Scripts that explicitly use smr file extensions (32-bit files) will need modification to work with smrx files. However, if a file with a .smr extension fails to open as a 32-bit file we try to open it as a 64-bit file, so a short term fix is to change the 64-bit file extensions to smr.

Feature	64-bit smrx file	32-bit smr file
Maximum file length in clock ticks	More than 10^{18}	2×10^9
Maximum file length with 1 μ s tick	255 thousand years	35 minutes 47 seconds
Maximum file size in bytes	16 EB (16 million TB)	2 GB or 1 TB (big file)
Maximum possible channels in a file (Spike2 allows 400)	65534	451
Time to find data item in a very large file is proportional to	Log(file size)	file size
Overhead in bytes for each gap in a waveform channel	16	0-32746, average 16374

This table compares some of the features of the two filing systems. The 64-bit system is designed with future developments in mind, the idea being that it is the basis of a format that can remain stable for many years. Particular emphasis has been placed on coping with very large files and long run times.

Hardware required

The minimum supported system for Spike2 version 8 is a computer running Windows XP SP3 with 1 GB of memory. Nowadays, almost any new desktop PC will have a suitable specification. If you run Vista or later you should have at least 2 GB of memory and ideally a multi-core processor. The more powerful the processor and the more memory your system has, the better Spike2 runs. Spike2 will run in Windows 8 as a desktop application. If your CPU is 32-bit, it must support the SSE2 instruction set.

To sample data, you need a CED Power1401 or a Micro1401 (mk II or -3). See the *Owners Handbook* that came with your 1401 for hardware installation instructions. Spike2 comes with all required 1401 drivers and will install the Try1401 test program to verify correct 1401 operation.

File icons



The various file types in Spike2 have similar icons so that you can recognise them in directory listings. This is the Spike2 application icon that you double-click to launch Spike2.



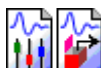
The icon on the left is for Spike2 data files. The icon on the right is for saved result views. The central icon is for an XY file. If you double-click one of these it will launch the Spike2 application (if it is not already running) and open the file.



Spike2 can output sequences of pulses, sine waves and voltage levels as it samples data. Output sequence files have this icon.



This icon is for Spike2 script files. A Spike2 script can automate data capture and analysis operations and extend the capabilities of the Spike2 program.



The icon on the left is for CED configuration files; these hold all the information needed to sample a new data file. The icon on the right is for CED resource files; these are usually associated with a data file and control how it is displayed.

Direct access to the raw data

Some users may wish to write their own applications that manipulate the Spike2 data files directly. A C library *Spike2: Son data storage library* is available from CED together with documentation sufficient for an experienced C programmer to use it. This library is for the 32-bit library, as used up to version 7 of Spike2. The library documentation is also available as a pdf file on the Spike2 distribution CD. To install it, select Custom Install and check the Additional documentation box.

The library used by Spike2 version 8 is written in C++ and includes the old 32-bit library. The interface has similar functions (in most cases there is a one to one mapping between the old functions and the new), but it is written taking advantage of the features of C++ and can be used to manipulate both the old and new files. Documentation for it is a work in progress.

Printed manuals

From Spike2 version 8 we no longer provide printed manuals. This is to save paper, reduce shipping costs and to reflect the fact that few users read them as the on-line, context sensitive help is much easier to use. You can get help in Spike2 in most situations where the system is waiting for input from you by pressing the F1 key or clicking a Help button. There are two PDFs of the on-line help on the distribution disk. These are mechanically generated from the on-line help (that is, it is not hand-optimised for reading as a manual). One version, `spike8.pdf`, is intended for use in a PDF reader and includes highlighted links, the other, `spike8print.pdf` (the version you are reading), is intended for printing and does not include highlighted links (so there may be invitations to find more information "here" that make no sense).

On-line Help

There is on-line Help available in the program, usually activated by the F1 key or by clicking a Help button. The help is generally context-sensitive, which makes it easier to use than this manual (even if used in a PDF reader) as it will often open at exactly the information you require. The manual you are reading is mechanically generated from the on-line Help and although there are differences between the on-line help and this manual, the main emphasis has been on making the on-line help as useful as possible. The result is that sometimes, the order of information in this manual will suffer.

CED Software Licences

CED software is protected by both United Kingdom Copyright Law and International Treaty provisions. Unless you have purchased additional licences as described below, you are licensed to run one copy of the software. Each copy of the software is identified by a serial number that is displayed by the **Help** menu **About Spike2...** command. You may make archival copies of the software for the sole purpose of back up in case of damage to the original. You may install the software on more than one computer as long as there is **No Possibility** of it being used at one location while it is being used at another. If multiple simultaneous use is possible, you must purchase additional software licences.

Additional software licences

The original licensee of a CED software product can purchase additional licences to run multiple copies of the same software. CED does not supply additional software media. As these additional licences are at a substantially reduced price, there are limitations on their use:

1. The additional licences cannot be separated from the original software and are recorded at CED in the name of the original licensee.
2. All support for the software is expected to be through one nominated person, usually the original licensee.
3. The additional licensed copies are expected to be used on the same site and in the same building/laboratory and by people working within the same group.
4. When upgrades to the software become available that require payment, both the original licence and the additional licences must be upgraded together. If the upgrade price is date dependent, the date used is the date of purchase of the original licence. If some or all of the additional licences are no longer required, you can cancel the unwanted additional licences before the upgrade.
5. If you are the user of an additional licence and circumstances change such that you no longer meet the conditions for use of an additional licence, you may no longer use the software. In this case, with the agreement of the original licensee, it may be possible for you to purchase a full licence at a price that takes into account any monies paid for the additional licence. Contact CED to discuss your circumstances.
6. If you hold the original licence and you move, all licences are presumed to move with you unless you notify us that the software should be registered in the name of someone else.

Installation

Your installation CD is serialised to personalise it to you. Please do not allow others to install unlicensed copies of Spike2.

Just put the CD-ROM in the drive and it will start the installation. You can also run the installation manually by opening the folder Spike8 on the CD-ROM, then open the disk1 folder and run setup.exe.

During installation

You must select a suitable drive and folder for Spike2 and personalise your copy with your name and organisation. You can have multiple versions on the same system as long as they are in different folders. If you have a previous version on your system, install to a different folder. The installation program copies the Spike2 program plus help, demonstration, example and tutorial files. It also copies and installs all required 1401 support (device drivers and control panels). In rare cases you may need to install the drivers manually; the installation program will tell you if this is the case and give you detailed instructions. Your system may require a restart after installation to get all drivers up to date.

Custom install

To install without 1401 support, or to copy additional documentation, choose Custom installation.

After Installation

If you are new to Spike2, please work through the Getting Started tutorial. Where you go next depends on your requirements. The *Spike2 Training Course Manual* is more descriptive than this help file, which is organised

as reference material. However, it covers all versions of Spike2 and you will occasionally need to refer to this on-line Help for version 8 specific details.

Updating and removing Spike2

You can update your copy of Spike2 version 8 to the latest version 8 release from our Web site: www.ced.co.uk. You can only update a correctly installed and licensed copy of Spike2 version 8. There are full instructions for downloading the update on the Web site.

Once you have downloaded the Spike2 update, you will find that the update program is very similar to the original installation, except that you must already have a properly installed copy of Spike2 for Windows version 8 on your computer.

Updates will include both bug fixes and new features. We will notify you by email (if we know your email address) of new releases. You can also register for this service on our web site. To stop emails, reply to them and ask to be removed from the list.

Removing Spike2

To remove Spike2: open the system Control Panel, select Add/Remove Programs, select CED Spike2 version 8 and click Remove. This removes files installed with Spike2; you will not lose files you created.

Versions of Spike2

This manual describes Spike2 for Windows version 8. We have generated the following versions of “Spike2 for ...”, listed in more or less chronological order of release:

DOS	Data and output sequencer files are still readable with later versions of Spike2.
Macintosh 68k	Data files, scripts and output sequences are compatible with later versions.
Windows ver. 2	This version ran on Windows 3.1 and 3.11.
Macintosh PowerPC	This was equivalent to Spike2 for Windows version 2. The last Macintosh version.
Windows ver. 3	The last version to sample with the standard 1401.
Windows ver. 4-6	These versions of Spike2 do not support the standard 1401.
Windows ver. 7	The last version to sample with a 1401 <i>plus</i> or the original micro1401 (Micro1).
Windows ver. 8	The version described here. It uses a new filing system that removes many of the limitations of sample time and file size of the previous versions while still reading all previous file formats. It runs in Windows XP service pack 3 onwards.

2: Getting started with Spike2

Getting started with Spike2

This tutorial teaches you the basic operations that manipulate Spike2 data files. Spike2 is a large program with many features; this short tutorial will get you started finding your way around.

Demonstration script

There is a demonstration script supplied with Spike2, called `BaseDemo.s2s`, that will give you an introduction to the program and demonstrate some of the ways that you can interact with it.

Use the **Help** menu **Getting started...** command to load and run the `BaseDemo` script. If this command is disabled, it means that Spike2 could not locate the script; re-installing Spike2 should restore it. You can control the script by clicking buttons or using the keyboard. Please remember to click **OK** or press the keyboard `Enter` key to keep the demonstration moving. If you prefer to just watch the demonstration, start it and click the **Options** button. Set the **Auto OK** time to 2 (this is how long to wait before the demonstration moves on).

If you are running the demonstration version of Spike2, you will find that this script has already been loaded for you. You will only need to select it from the list when you use the **Script** menu **Run Script** command.

The demonstration script loops back to the start, so you will need to interrupt it to continue with the tutorial. Hit the **Exit!** button on the toolbar at the top of the screen to stop it running.

Opening a file to view

In this step you will open the demonstration data file that was shipped with Spike2. Follow these steps:

1. Open the **File** menu and select the **Open...** command
2. Navigate to `My Documents` folder and within that there should be a `Spike8` and within that `data`; open `data` and double click on `demo.smr` (if you have moved files since your installation you may have to search around to find this file, if all else fails, use any Spike2 file). Spike2 can read data files with the extensions `.smr` (an old format file with 32-bit times and a maximum size of 1 TB) and `.smrx` (a new format file with 64-bit times and a maximum size limited by the operating system).
3. A new window will open. Arrange the help window and this new window so that you can see both.
4. To follow the tutorial, you should see a window holding at least one data channel and with a horizontal scroll bar at the bottom. If this is not the case your file is in a mess!

Spike2 displays the file in the state in which it was last saved (as long as it can find a file with the same name and the extension `.s2rx` or `.s2r`). You are looking at the raw data in the file. We call this a time view because it displays a time history of the data and the axis at the bottom is in seconds.

Clean up a messy file

You can tidy up by following these instructions:

1. Click on the window titled `demo.smr`
2. From the **View** menu select the **Standard Display** option
3. If the fonts used seem too large or too small open the **View** menu **Font** dialog and select **Times New Roman, Regular 10 point**
4. If the colours are grotesque open the **View** menu and select the **Change Colours...** option and click **Reset** for a standard colour scheme or choose a better combination. As a last resort, the **View** menu **Use Black And White** converts everything into a black foreground and a white background.

Data channels

There are several *data channels* displayed in the window. These channels can hold different types of data: channel 1 holds a waveform, channels 2 and 3 holds events and channel 31 holds keyboard markers.

Selecting channels

You select channels by clicking on the channel number. The channel number is usually to the left of the channel, but can also be on the right if this is selected in the View menu Show/Hide dialog. If the channel number has also been hidden you can click the region a few pixels wide in the place where the channel number would be if it were displayed. Spike2 highlights the channel number. Hold down the `Shift` key and click on a channel to select all channels between it and the last selection. Hold down `Ctrl` to select discontinuous channels. Several commands work on a list of selected channels (for example y axis display optimisation).

Channel modified indicator

If the channel number is displayed in red, this means that the channel data has a Channel process or a Marker Filter applied to it or is not displaying the first marker code.

Zoom buttons

The bottom window edge holds four buttons and a scroll bar. The scroll bar controls movement through the file. If you resize the window, the same data is drawn, scaled to the window. The two buttons to the left of the scroll bar change the time range.



This button halves the time range (zoom in). The left edge of the display remains fixed. You can zoom in until the display is one time unit wide.



This button doubles the time range (zoom out). The left edge of the window does not move unless the start plus the new width exceeds the length of the file. If the new width exceeds the file length, the entire file is displayed.

X axis short cut keys

The following short cut keys combinations can be used to navigate the x axis:

Key	Operation
Left arrow	Scroll 1 pixel left.
Right arrow	Scroll 1 pixel right.
Shift+Left	Scroll several pixels left.
Shift+Right	Scroll several pixels right.
Ctrl+Left	Scroll half a screen left.
Ctrl+Right	Scroll half a screen right.
Ctrl+Shift+Left	Time views only. If there are event channels selected, search all selected channels for the previous event that is nearest to the centre of the screen and make it the centre of the screen or make a sound if there are no selected channels or no more events.
Ctrl+Shift+Right	Time views only. If there are event channels selected, search all selected channels for the next event that is nearest to the centre of the screen and make it the centre of the screen or make a sound if there are no selected channels or no more events.
End	Scroll to the end of the data.
Home	Scroll to the start of the data.
Ctrl+E	Expand (zoom out) the data view around the left edge of the window.
Ctrl+R	Reduce (zoom in) the data view around the left edge of the window.
Ctrl+U	Expand (zoom out/Up) the data view around the centre of the window.
Ctrl+I	Reduce (zoom In) the data view around the centre of the window.

The full list of keyboard commands in a data view can be found in the General information chapter.

Cursor buttons



This button adds a vertical cursor to the display (up to 10 vertical cursors can be present in a window). A cursor is a dashed line used to mark positions. You remove cursors with the Cursor menu Delete option. You can add vertical cursors in five ways.

1. Click and release the button to add a cursor in the centre of the window.
2. The Cursor menu New cursor command adds a cursor in the centre of the window.
3. Click and release the right mouse button at the position where you want the new cursor. When the context menu appears, select New Cursor and a cursor will appear at the mouse click position.
4. The short cut keys `Ctrl+0`, `Ctrl+1` through `Ctrl+9` place cursors 0 to 9 in the centre of the window.
5. From a script you can use the `CursorNew()` or `CursorSet()` commands to create cursors. Script users have more control over cursors and can set custom cursor labels.

You can also use a short cut key to scroll the x axis to locate a cursor. `Ctrl+Shift+0` through `Ctrl+Shift+9` will scroll so that the selected cursor is as near to the centre of the window as possible. You can read more about vertical cursors in the Cursor menu chapter.



There are also horizontal cursors. These are most useful as part of a script, but you can create them interactively from the cursor menu or from the right mouse click context menu or by clicking the horizontal cursor button. You can read more about horizontal cursors in the Cursor menu chapter.

Cursor style and pointers

Click on the cursor button so that at least one vertical cursor is visible. The mouse pointer changes over a cursor or a cursor label:



This indicates that you can drag the cursor. If you drag beyond the window edge, the window scrolls. The further beyond the edge, the faster the scroll. Dragging hides the label unless the `Ctrl` key is down or you drag the label. If a horizontal cursor is outside the range of the y axis of a channel, you can still see the cursor label and even drag it sideways. Hold down `Ctrl` and click and drag to bring it back into the channel area.



If you position the mouse pointer over the cursor label, the pointer changes to a 4-headed arrow to indicate that you can drag both the cursor and the label. This can be useful when preparing an image for publication and you need the cursor label to be clear of data. If you move the pointer to one side, or hold down the shift key, the pointer becomes a two-headed vertical arrow and you can drag the label, but not the cursor. If a cursor is off the edge of the display area and you can still see the label, hold down `Ctrl` and click and drag to move it back into view.

There are four labelling styles for the cursor: no label, position, position and cursor number, and number alone. You select the style with the Cursor menu Label mode option.

Active cursor modes

Type `Ctrl+0` to centre cursor 0 on the screen. Drag any other cursors away from cursor 0. Now right click on cursor 0 and select **Cursor 0->Active mode...** from the context menu to open the Cursor mode dialog.

All vertical cursors can be *Static* or *Active*. A static cursor stays where you leave it. An active cursor can reposition itself by searching for user-defined data features.

Set cursor 0 to **Peak find** on channel 1. Set **Amplitude** to 0.1, **Minimum Step** to 0 and click the OK button. Now try the `Ctrl+Shift+Right` key combination (**Right** is the right arrow key). Each time you press these keys, cursor 0 seeks the next peak on channel 1 that is at least 0.1 y axis units high. `Ctrl+Shift+Left` moves cursor 0 in the opposite direction. You can set any vertical cursor into active mode. When cursor 0 moves, any other active cursors apply their search method in rising cursor number order.

There are a wide variety of search methods that can be used to locate data features. You can read more about active cursors in the Cursor menu active cursors description, or by pressing the `F1` key when the Cursor mode dialog is open. For now, leave cursor 0 in **Peak search** mode and move on to the next step.

Automatic measurements to XY view

Now use the Analysis menu Measurements->XY view command to open the Settings for XYPlot dialog.

This dialog lets us take a set of measurements over a time range and generate a graph which you can print, save or copy as either a picture or as a table into other applications. It is likely that the dialog settings will be suitable for this demonstration, but check that the Cursor 0 stepping region holds: Channel=1 Sinewave, Method=Peak find, Amplitude=0.1 and Minimum step = 0. Make sure that the Ignore cursor step if field is blank and that the User check positions box is not checked.

Check in X Measurements that: Type=Time at Point, Time=Cursor(0). Check in Y Measurements that: Type = Value at Point, Channel=1 Sinewave, Time=Cursor(0), Width=0.

Finally check that Points=0 and then click the New button. The Process XYPlot dialog opens so that you can set the region of the data file to analyse. It should hold a sensible start and end time range, so just click the Process button.

Spike2 will generate a new window that displays a graph of the x and y measurements. In this case you will get a graph of peak amplitudes against time in seconds. You can adjust the appearance of the new XY view (remove joining lines, change the markers used for each data point, and the like), but we are here for a quick tour, so close the XY view (click in the X box at the top right).

To find out more about the measurement system you can read more in the Analysis menu Measurements description or press the F1 key in the Settings for XYPlot dialog.

Zoom in on an area

Move the mouse pointer to the waveform channel. Click the left mouse button and drag a rectangle round a waveform feature and release the button. The window displays the area within the dragged rectangle. If the rectangle covers more than one channel, only the time axis changes. If your rectangle lies within a channel and has zero width, only the y (vertical) axis changes.



The mouse pointer changes to a magnifying glass when you hold the left mouse button down in the data channel area to show that you are about to magnify the data.



If you hold down the `Ctrl` key and left click, the mouse pointer is the un-magnify symbol and the rectangle holding the channel area shrinks to the rectangle you drag

Whichever method you use to scale the data, you can return to the previous display using the Edit menu Undo command or the keyboard short-cut `Ctrl+Z`. If you release the mouse button with the pointer in the same position from which you started the drag, the display does not change.

On screen measurements

If you hold down the `Alt` key before you click and drag, Spike2 displays the size of the dragged rectangle next to the mouse pointer and does not zoom the display. With the mouse button held down, release `Alt`, then use the `C` key to copy the current measurement to the clipboard or the `L` key to copy it to the Log view.



Zoom a channel

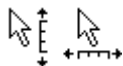
You can zoom one channel to use the entire display area!

Double click anywhere in the waveform channel. The channel will expand to occupy the entire window.

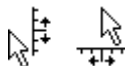
Double click in the waveform channel again and your previous display is restored.

You can also change the display to show a channel and all duplicates of it by holding down the `Ctrl` key, then double-clicking on a channel. This is useful with sorted spikes, where duplicates of a channel are used to display spike classes.

Using x and y axes to scroll and zoom



When the cursor is over the tick marks of an axis, you can drag the axis. This maintains the current axis scaling and the window moves to keep pace with the mouse pointer. You can do this with most x and y axes in Spike2. This is particularly useful for y axes as they do not have a vertical scroll bar. The window does not update until you release the mouse button. If you hold down the **Ctrl** key, the window will update continuously.



When the cursor is over the axis numbers, a click and drag changes the axis scaling. The effect depends on the position of zero on the axis. If the zero point is visible, the scaling is done around the zero point; the zero point is fixed and you drag the point you clicked towards or away from zero. If the zero point is not visible, the fixed point is the middle of the axis and you drag the point you clicked towards and away from the middle of the axis

In a time view, result view, or XY view, you can drag the y axis so as to invert the axis. You can prevent this happening by setting an option in the Edit Preferences command Display tab. You are not allowed to invert the X axis. You are not allowed to invert or scroll the y axis in the spike shape dialogs.

X Range dialog




Click this toolbar button or double-click the time (x) axis of the display to open the X Range dialog. Experiment with the time axis. You can type new positions or use the pop-up menus next to each field to access cursor positions and the maximum time in the file. The most important dialog controls are:

Units	In a time view you can choose between an x axis in seconds, in hours, minutes and seconds and as time of day. Time of day works for files sampled with versions of Spike2 from 4.03 onwards that save the start of sampling time in the file.
Left	Sets the window start time. You can enter times as seconds, for example 3665.2, or in a time format as 61:05.2 (61 minutes, 5.2 seconds) or as 1:01:05.2 (1 hour, 1 minute, 5.2 seconds). The time format extends up to days, so 1:1:1:1 is 1 day, 1 hour, 1 minute and 1 second. At the moment, times are relative to the start of the file, even with time of day mode selected. In addition to typing times, or selecting a time from the drop-down list, you can type in expressions using the maths symbols + (add), - (subtract), * (multiply) and / (divide). You can also use round brackets. For example, to display from 1 second before cursor 1 to one second past cursor 1 set Left to <code>Cursor(1)-1</code> and Right to <code>Cursor(1)+1</code> . The Draw button is disabled if you type an invalid expression, or if the Right value is less than or equal to the Left value or if the new range is the same as the current range.
Right	Sets the window end time using the same format as the Left field.
Width	Shows the window width. You can either set the left and right positions, or the left position and the width. Check the box to keep the width the same when the Left field changes.
Show All	Expands the time axis to display all the file and closes the dialog.
Draw	If the axis range had been edited, use this to redraw the data to match the new range.
Close	This closes the dialog; it does not update the axis range.
Cancel	This undoes all changes made with the dialog and closes it.

The **Large tick spacing** and **Tick subdivisions** fields let you customise the axis. Values that would produce an illegible axis are ignored. Changes to these fields cause the axis to change immediately; you do not need to click **Draw**. See the X Axis Range dialog documentation for full details of all dialog items.

Drop-down menus

A drop-down menu item is marked in a dialog by a triangle pointing downwards () that displays a list of possible values for the field when you click the mouse on it. You can select one of the items in the list, or in some cases you can type in your own value.

Y Range dialog



Click this toolbar button or double-click on the y axis of the waveform channel to open the Y Range dialog. This dialog changes the y axis range of one or more channels. The **Top** and **Bottom** fields set the values to display at the ends of the axis. Experiment with the y axis range.

- | | |
|-----------------|--|
| Channel | A pop-up menu from which you can select any channel with a y axis, or all channels with y axes, or all selected channels. |
| Optimise | This button changes the y axis ranges of the set channel(s) to fit the data and closes the dialog. You can optimise the y axis without opening this dialog. The Ctrl+Q key combination optimises all selected channels, or all channels if no channel is selected. |
| Show All | This shows the full range of a waveform channel and closes the dialog. Sampled waveform data is stored on disk as 16-bit integers with a numeric range of -32768 to 32767. The y axis is set to display the full range. For other channels it will display a sensible range. |
| Draw | If the axis range had been edited or the channel locking fields have changed, use this to redraw the data to match the new settings. It does not close the dialog. |
| Close | This closes the dialog; it does not update the axis range. |
| Cancel | This undoes all changes made with the dialog and closes it. |

The **Lock axes** and **Group offset** fields are hidden unless you have selected a channel that is part of an overdrawn group. These fields are disabled unless the current channel is the first of a group. If you check the **Lock axes** box, all the grouped channels use the y axis of the first channel in the group to set their display range. The **Group offset** field sets a vertical display offset, in y axis units, to apply between channels in the group. You would use this if you wanted to draw many channels with the same mean level on the same axis and wanted to separate the channels vertically. The offset only applies to the visual display, not to any measurement made on the data.

The **Large tick spacing** and **Tick subdivisions** fields customise the axis. Values that would produce an illegible axis are ignored. Changes to these fields are applied immediately; you do not need to click **Draw**.

If no channel has a y axis, open this dialog from the View menu. See the Y Axis Range dialog for full details.

Channel draw mode dialog

Data files hold two basic channel types: waveform and event. Waveform channels hold a list of values representing the waveform amplitude at successive time intervals. Event channels hold the times at which something happened (and more data, depending on the channel type).

Open the View menu Channel Draw Mode dialog. Experiment with different drawing modes for channel 3. Click **Draw** to update the display without closing the dialog. Click **OK** to close the dialog.

Show and Hide channels

Open the View menu Show/Hide Channel dialog. This sets the channels to display in your window. With up to 100 channels in a file plus memory channels and duplicate channels, this ability is quite important if you are to see any detail!

The list on the left of the dialog holds all the channels that can be displayed. Check the box next to a channel to include it in the display list. You can also turn all channels on and off with the buttons.

You can also show and hide the channel number, axes, background grid and the horizontal scroll bar in the window from this dialog and position the y axis on the right and choose to draw axes as scale bars.

Click the **Draw** button to see the result of your changes without closing the dialog, or click **OK** to close the dialog and see the changes. If you make a complete mess of your window you can use the View menu Standard Display command to clean things up.

Channel order

Make sure that the demo file is the current window and use the View menu Standard Display command to tidy things up. Click on the Keyboard channel number (31) and drag it down over the other channel numbers.

As the mouse pointer passes over each channel, a horizontal line appears above or below the channel. This horizontal line shows where the selected channel will be dropped. Drag until you have a horizontal line below channel 1 and release the mouse button. Channel 31 will now move to the bottom of the channel list. Type `Ctrl+Z` or use the Edit menu Undo to remove your change.

You can move more than one channel at a time. Spike2 moves all the channels that are selected when you start the drag operation. For example, hold down `Ctrl` and click on the channel 3 number. Keep `Ctrl` down and click and drag the channel 2 number. When you release, both channels will move. The mouse pointer shows a tick when you are in a position where dropping will work.

The usual Spike2 channel order is with low numbers at the bottom of the screen. If you prefer low numbers at the top of the screen, open the Edit menu Preferences and check Standard Display shows lowest numbered channel at the top, then use the View menu Standard Display command.

Channel spacing

Change the channel 3 drawing mode to Mean frequency. Hold down the `Shift` key and move the mouse over the data area. Hold the `Shift` key down and click. Drag up and down and release the mouse.

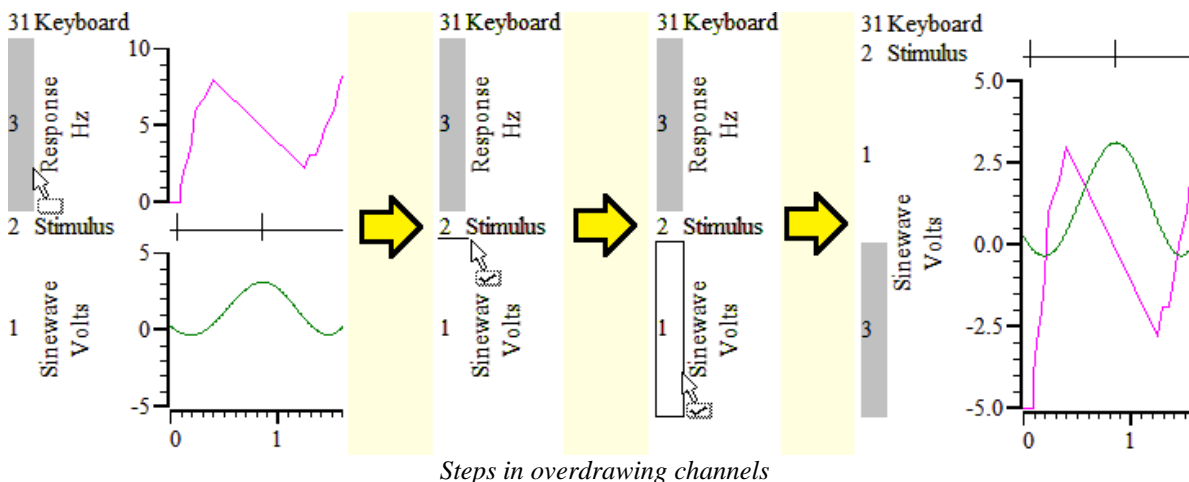
When you click with `Shift` down, the mouse jumps to the nearest channel boundary and you can change the boundary position by dragging. With `Shift` down, you can move the edge up and down as far as the next channel edge. You can undo changes or use Standard Display to restore normal sizes.

If you add `Ctrl`, all channels with a Y axis are scaled. If there are no channels with a Y axis above or below the drag point, then all channels scale. You can force all channels to scale by lifting your finger off the `Shift` key (leaving `Ctrl` down) after you start to drag the boundary.

Channel overdraw

You can overlay channels of data in a time view or a result view. This is done interactively by dragging channel numbers on top of each other or from a script by using the `ChanOrder()` command. To do this in the `demo.smr` example file:

Click the "3" of channel 3 and drag it on top of Channel 1 and release.



Channels 1 and 3 now share the same space with the channel numbers stacked up next to the Y axis. The visible Y axis is for the top channel number in the stack. You can promote another channel in the stack to the top by double-clicking the channel number. The channels retain their own y axes and scaling. You can remove a channel by dragging the channel number to a new position.

When you drag channels, and at least one of the selected channels has a Y axis, you can drop the channels with a Y axis on top of another channel with a Y axis. As you drag, a hollow rectangle appears around suitable dropping zones. You can also drop between channels when a horizontal line appears.

Merged channels are drawn such that the channel with the visible Y axis is drawn last. If you have a channel that fills in areas, such as a sonogram or an event channel drawn as rate mode, put is at the bottom of the stack, as it will mask channels below it in the stack. If overdrawn channels have been given their own channel primary colours, the channel numbers for the overdrawn channels are displayed in the channel primary colour.

Use the View menu **Standard Display** command to tidy things up before you continue.

Cursor values



Make sure there are cursors in the window, then click this toolbar button or use the **Cursor** menu **Display Y Values** option to open the **Cursor Values** dialog. The columns show the cursor times and values. For channels without a y axis, the value is the next event time after the cursor. If you move cursor or alter a channel display mode, the values change. If you add or remove channels in the display, they are added or removed in the dialog.

The **Time zero** and **Y zero** check-boxes select relative rather than absolute measurements; the radio buttons set the reference cursor. Reference cursor values are unchanged; values at other cursors have the reference value subtracted.

You can copy dialog fields to the clipboard. Click and drag for multiple selections. Click a top or left hand cell to select an entire column or row. Hold down **Ctrl** to make non-contiguous row or column selections.

Cursor regions



Click this toolbar button or use the **Cursor** menu **Cursor Regions** command. Experiment with changing cursor positions and channel display types. The regions dialog looks at the data values between cursors. There are several modes set by the drop down list at the bottom of the dialog:

Area	The area between a waveform and the y axis 0.0 level or the number of events in the region.
Mean	The average level of a waveform, or the number of events divided by the region width.
Slope	The gradient of the least-squares best fit line to the waveform, no meaning for events.
Sum	The sum of waveform values between the cursors or the number of events in the region.
Area(scaled)	This mode is the same as Area , except that when a Zero region is set, the value in the zero column is scaled to allow for the relative column widths before being subtracted.
Curve area	Each data point makes a contribution to the area of its amplitude above a line joining the end points multiplied by the x axis distance between the data points.
Modulus	Each data point makes a contribution to the area of its absolute amplitude value multiplied by the x axis difference between data points. This is equivalent to rectifying the data, then measuring the area. If a zero region is specified, the amount subtracted from the other regions is scaled by the relative width of the regions.
Maximum	The maximum value found between the cursors.
Minimum	The minimum value found between the cursors.
Peak to Peak	The difference between maximum and minimum values found between the cursors.
SD	The standard deviation from the mean of the values between the cursors. If there are no values between the cursors the field is blank.
RMS	The value shown is the RMS level of the values found between the cursors. If there are no values between the cursors the field is blank.
Extreme	The value shown is the maximum absolute value found between the cursors. Thus if the maximum value was +1, and the minimum value was -1.5, then this mode would display 1.5.
Peak	The value shown is the maximum value found between the cursors measured relative to a baseline formed by joining the two points where the cursors cross the data.
Trough	The value shown is the minimum value found between the cursors measured relative to a baseline formed by joining the two points where the cursors cross the data.

You can also make relative measurements by checking the **Zero region** box and choosing a reference region.

Result views


Windows holding raw, unprocessed data are called *Time views*. There is another type of data window, called a *Result view*, that holds the result of analysing time view data. These windows hold one or more channels of data, each channel has the same number of data points and x axis are suitable for waveforms or histograms. Channels can also have event times associated with them for raster displays. There are two steps in the analysis:

1. You set the type of analysis, the channels to analyse, the number of points or bins to generate and any other parameters required. This creates a new, empty result window.
2. You set a time region of the time window and Spike2 calculates the result and adds it to the result view.

You may repeat step 2 as many times as is required to accumulate results from different sections of data. The new window behaves like a time view containing a single channel of data. Result views can also be created from a script.

Make an interval histogram

Close all windows except the original time view of `demo.smr`. Then:

1.  Click this button on the toolbar or use the Analysis menu New Result View command to select Interval Histogram.
2. Set the channel to analyse. The channel list in the pop-up menu includes event channels only. Channel 3 of the `demo` file is the best for this example.
3. Set the number of bins in the histogram, we suggest 150.
4. Set the width of each bin, in seconds. Bins must be at least one Spike2 clock tick wide. The bin width is rounded to a multiple of this clock tick. We suggest 0.01 seconds (this is 10 milliseconds).
5. Leave the last field set to 0.0 seconds. This is the minimum interval that appears in the new window.
6. Click the **New** button to create a new window and open the **Process** dialog.

Process dialog

Now set the region of the data file to analyse:

1. Set the start time for analysis to 0
2. Set the end time to `View(-1).MaxTime()` (this is the time of the last data item in the file), this is in the pop-up menu.
3. Check the **Optimise Y axis after process** box to scale the y axis to the data automatically.
4. Click **Process** to analyse the data and display the result.

The **Clear result view before process** check box sets the result window contents to zero before you analyse the data, otherwise each new result is added to the previous one. The **Settings** button takes you back to the previous step.

Repeating and extending a process

You can add more data into the result or change the progress settings and process again:

1. Recall the process dialog by selecting the **Process** command from the **Analysis** menu.
2. Click the **Process** button again. The data in the result window will double in size (as long as you have not checked the **Clear result view before process** box).
3. Recall the process settings dialog by selecting the **Process settings** command from the **Analysis** menu.
4. Change the number of bins to 400, and the time resolution to 0.004 then click the **Change** button and confirm that you wish to continue.
5. Select a region for processing with the new settings.

Result view drawing modes

Experiment with the Channel Draw Mode command in the View menu.

There are four different drawing styles available for result windows: Histogram, Line, Dots and SkyLine. These styles are self-explanatory. The various analysis routines that create result windows will select an appropriate style.

See the View menu Result view drawing modes documentation for more detailed information.

Result view cursors

Experiment with cursors in this new window.

You will find that the cursors behave in a very similar manner to the original time window and that you can use the Cursor menu Cursor Values and Cursor Regions commands in the same way.

Other sources of information

If you have worked through the Getting started tutorial, you have the basic skills required to make use of Spike2 for interactive data analysis. You can find more detailed information in the following sections:

<i>Sampling data</i>	This describes the different data types Spike2 supports and how to configure Spike2 to sample them. It also covers arbitrary waveform output (output through the 1401 DACs of previously sampled waveforms or waveforms generated by the script language).
<i>Data output during sampling</i>	This describes the output sequencer, which allows you to produce precisely timed digital pulses, voltage ramps and steps and cosine waves. You can also detect changes in external signals in real time and respond to them in much less than a millisecond.
<i>File menu</i>	This describes all the commands in the File menu: Opening and saving files and configurations, exporting data from Spike2 and Printing.
<i>Edit menu</i>	This describes the commands in the Edit menu including the Preferences (well worth a visit).
<i>View menu</i>	This describes the View menu, which covers all aspects of what you see in each data window.
<i>Analysis menu</i>	This describes how you can generate result views from time view data, how to take measurements to a new XY view, how you generate Memory channel (for example by picking peaks from waveforms), the marker filter, spike shape creation and editing and the digital filter.
<i>Window menu</i>	Describes the commands in the window menu.
<i>Cursor menu</i>	Covers the use of vertical and horizontal cursors, how to set active cursors and how to use the Cursor regions and Cursor values dialogs.
<i>Sample menu</i>	Describes the commands in the sampling menu including off-line waveform output and the use of TextMark data during sampling.
<i>Script menu</i>	This is a brief overview of the commands in the script menu. There is a separate manual <i>The Spike2 script language</i> which has a detailed description of the language. You should also see the <i>Spike2 Training Course Manual</i> for more details of selected script topics.
<i>Spike shapes</i>	This chapter has a more detailed explanation of the use of Spike2 to sample spike shapes and the use of the template system to identify spike shapes.
<i>Digital filtering</i>	This describes the digital filter dialog in detail, and also has a more technical section which discusses the use of digital filters.

<i>XY views</i>	This has a brief introduction to XY views from the point of view of a script writer and introduces a script that can generate "waterfall" displays.
<i>Signal Conditioners</i>	Spike2 can control programmable signal conditioners such as the CED 1902, the Axon CyberAmp and the Power1401 with the ADC gain option fitted.
<i>Utilities</i>	There are additional programs provided with Spike2 to fix damaged data files and to check out your 1401.

To learn about using the script language for analysis you should read about the script language and investigate the example scripts provided with Spike2:

<i>Script language Introduction</i>	Basic introduction to the script language including a "Hello world" script and how to record simple actions and then edit them into a more useful script.
<i>Script window and debugging</i>	Working with the script window and the associated script commands and how to use the built-in script debugging system.
<i>Script language syntax</i>	An explanation of the script language structure, keywords, variable types, data arrays and basic programming structures together with simple programming examples.
<i>Functions by topic</i>	Spike2 has more than 500 built-in script functions. This section groups them together by function. For example, if you want to find all the script functions that relate to sampling, then start here.
<i>Alphabetic function list</i>	This is the full list of all built-in functions arranged alphabetically. Each function comes with a description and some also include examples of use.

The *Spike2 Training Course Manual* is another resource you can use. It contains getting started chapters on many topics, and is an invaluable reference for the script writer as it has worked examples that cover many common requirements. If you like video tutorials, you can find several on our web site that demonstrate specific features in detail.

3: General information

General information

This covers topics that are important throughout Spike2 and miscellaneous items that don't fit anywhere else.

Spike2 clock tick

All items in a Spike2 data file exist at an integer multiple of a basic time unit, the underlying *clock tick* or *time resolution* for the file. This unit is typically very small, and usually lies between 1 and 100 microseconds. A 64-bit `smrx` data file can run for 8×10^{18} of these clock ticks, which at 1 microsecond per tick would be 255 thousand years. Put another way, if you use a 64-bit `smrx` data file, the length of the file is not a consideration and you should probably run at 1 microsecond per tick unless there is a pressing reason to run slower.

However, if you use a 32-bit `smr` data file, it can only run for a little more than 2 billion (2×10^9) of these clock ticks. At a clock tick of 1 microsecond, a file is limited to around 36 minutes. A 10 microsecond clock tick file can run for almost 6 hours. At 100 microseconds the file could run for more than 2 days.

The value of the clock tick is usually some multiple of microseconds or tenths of microseconds when the original source of the data is a CED 1401 data acquisition unit. However, when data is imported or a data file is created by a script, the clock tick could be any value.

The clock tick used when sampling is set in the **Resolution** tab of the **Sampling configuration** dialog. From the script language, the clock tick is set by the `FileNew()` command when creating a file. The `FileTimeBase()` script command can be used to read the clock tick and it can also be changed to adjust the time base of a file (for instance when a EEG data is played back from a recorder at several times the normal rate).

Data file versions

Spike2 data is stored in files with the extensions `.smr` (for the original 32-bit times format) and `.smrx` (for the 64-bit times format). The 32-bit file format was designed around 1987 and has been much extended since. There were 9 major version changes to the 32-bit format; we are still on the first version of the 64-bit format. We have always made sure that more modern versions of Spike2 will read older file versions. Unless you are interested in history, or you have an old version of Spike2 and cannot read a data file, you can stop reading this page now.

If you are interested in the technical details of the library, or need to interface to it, there is documentation and header files on the CED web site (follow links to **Downloads** and then **Son library**).

Version Change history

- 1 The original SON filing system, written in Pascal, supported waveform and `Event` data only.
- 2 New `Marker` data type. Added extra space to the file header for future expansion. The library was much faster reading large data files as it remembered the last accessed data.
- 3 Added the `FilterMarker` function. Changed the meaning of the waveform channel divide to prevent an apparent change of sampling rate if a waveform channel was added or deleted.
- 4 Added the `AdcMark` data type. Changed the use of `FilterMarker`. The library now caches the current data block to avoid re-reading when the required block is already in memory.
- 5 Added the `TextMark` and `RealMark` data types and the `MaxTime` and `ChanMaxTime` functions. C library versions written for the IBM PC in DOS and Windows and for the Macintosh. The version 5 libraries write version 3 and 4 files if the data does not need new features.

In 1998, we extended the C version to support read-only files and to allow more data to be written per call. We added lookup tables to speed up data access in long files and write buffering to remove the need for `FastWrite`, allow peri-triggered sampling, and speed up writing).

- 6 Documented for C/C++ use only with information on use as a DLL by other languages. Added the `RealWave` channel type. `AdcMark` channels have multiple traces. Waveform rate divider is now 32-bits. Added support for time and date stamp, basic time unit and an application identifier. Files still compatible with versions 3, 4 or 5 if they do not use version 6 features.

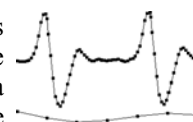
- 7 You can choose to round the sizes of `AdcMark` and `TextMark` extended marker types up to a multiple of 4 bytes so that we can build the library on systems that insist on aligned data access.
- 8 Altered storage of channel numbers to avoid the bit used to hold the initial state for a level channel, allowing the maximum number of channels to be increased to 451. Extended the lookup table so that the table for a channel starts small and grows up to a limit (larger than the old fixed size).
- 9 Added support for big files (up to 1 TB), rewrote the internal lookup table system to improve speed and saves the table as part of the file in Big file mode. Macintosh (big-endian format) support was removed. Linux support is added. In a version 9 file, pointers to disk space that were previously byte offsets (but multiples of 512) are now block pointers (a pointer value 10 means offset 5120). This increases the maximum file size by a factor of 512. There are changes to the file header to support a lookup table on disk and to channel headers to track the block counts. There were also type changes to allow compiling as 64-bit code on Windows and Linux.
- 256 **64-bit times.** A complete redesign of the filing system to remove the time and disk space limitations of the original while leaving a system that is similar enough to the original to store the same data and behave in similar ways. Times are stored as 64-bits and the files can theoretically be of any size up to 2 to the power 64 bytes (around 10 to the power 19 bytes).
0x100
- Practical considerations, such as the need to copy files in reasonable time will limit file sizes to a small number of TB for the next few years. Times are stored to 64-bit accuracy, but some operations within Spike2 use floating point numbers to manipulate file times, and a floating point number has some 53 bits of precision, so this also limits the available maximum time. However, with 1 microsecond ticks, you can still sample for tens of years before this becomes an issue.
- You can access both 32-bit and 64-bit files using the same API, so although there is software effort required to use the new format, once done you can read both old and new formats with the same code. Also, if the `son64.dll` file is placed in the same folder as a recent `son32.dll` file, old code that reads old 32-bit files can also read a new file (as far as the first 32-bits of clock ticks).

Data channel types

In a Spike2 time view there are two basic data types: Waveforms and Events. These are then further subdivided into more types.

Waveform and RealWave

Waveforms are stored as a list of data values that represent a signal amplitude. The time gaps between the samples are all the same, and this is known as the *sampling interval*. The reciprocal of the sampling interval is known as the *sampling rate*. So a signal with a sampling interval of 0.01 seconds (or 10 milliseconds) has a sampling rate of 100 Hz. The sampling interval must be an integer multiple of the clock tick for the file. For example, if the underlying clock tick was 10 microseconds, waveform sample intervals of 10, 20, 30, 40... $n \times 10$ microseconds are possible. Put another way, sampling rates of 100 kHz, 50 kHz, 33.3 kHz, 25 kHz... $100/n$ kHz are possible.



Spike2 stores waveforms on disk as either 16-bit integers in the range -32768 to 32767 (a Waveform channel) or 32-bit floating point values (a RealWave channel). Both these channel types have an associated *scale* and *offset* value plus a text string that defines the user units to use for the y axis when displaying the data. For a Waveform (integer) channel, the scale and offset are used to convert the stored integer values into user units. This is arranged so that a scale value of 1.0 and an offset value of 0.0 map the input of a 1401 device with a ± 5 Volt range into Volts. That is:

$$\text{user units} = \text{integer value} * \text{scale} / 6553.6 + \text{offset}$$

RealWave channels are stored in user units. They use the scale and offset values whenever a RealWave channel needs to be converted to an integer value for use as a Waveform. In this case:

$$\text{integer value} = (\text{user units} - \text{offset}) * 6553.6 / \text{scale}$$

If the result of this would exceed 16-bit integer range (-32768 to 32767), it is limited to -32768 or 32767.

Waveform and RealWave channels can have gaps in them where there is no data. The very first data point of the channel determines the possible positions of all subsequent points as all points lie at a time that is:

$$\text{time of first point} + n * \text{sampling interval}$$

where n is an integer value. You can read about sampling waveform data in the *Sampling data* chapter.

Event

The basic concept of an event is that it is a time stamp. The time stamp is a 64-bit number that is the count of the number of underlying clock ticks from the start of the file to the event. The time of the event in seconds is:



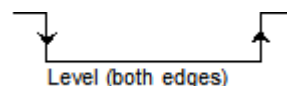
$$\text{time in seconds} = \text{timestamp} * \text{underlying clock tick period}$$

Spike2 has two event types that are exactly equivalent: Event+ and Event-. The + and - relate to how the data was captured. The plus means that the time was from a signal rising through a trigger level, the minus means that the signal was falling through a level. This is only important when data is captured. You can read about sampling event data in the *Sampling data* chapter.



Level

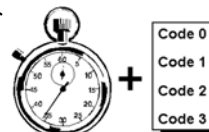
There is a third simple event type, Level. In the 32-bit filing system, this was stored in exactly the same way as Event data and each data block had a flag that indicated the direction of the first edge in the block. This had the advantage of being compact, but the disadvantage that deleting an edge caused all following edges (but only in the block) to invert and caused all kinds of logical problems. The new son64 system implements Level channels as Marker data using marker code 0 for a low-going edge and code 1 for a high-going edge.



When stored in a memory channel, we still treat Level data as a list of times to make it easy to insert and delete edges and to maintain backwards compatibility with Spike2 version 7. However, when we write Level data to a son64 file we save it in Marker format.

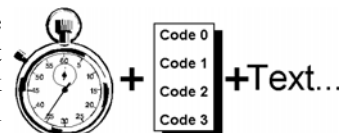
Marker

Spike2 extends the event concept by attaching data to each event. The simplest form of attached data is the Marker, which adds 4 8-bit marker codes to each event time. These codes can then be used to filter the event data. For example, the keyboard marker channel that Spike2 always adds to a sampled data file stores the keys pressed during data capture in the first marker code. If you associate a particular key with a recurrent event during sampling, you can filter the channel to show only events with that particular key code. As a Marker is exactly like an event, anything that you can do with an event channel can be done with an (optionally filtered) Marker channel. You can read about sampling Marker data in the *Sampling data* chapter. You can also create such channels as memory channels and then save them to make a permanent channel.



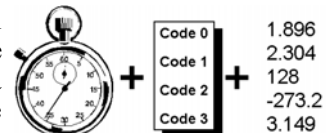
TextMark

A TextMark channel is exactly the same as a Marker (so anything that can be done with a Marker can be done with a TextMark), except that it has a text string attached. There is a maximum size of string set for each TextMark channel. You can set a TextMark channel for use during sampling. You can also create such channels as memory channels and then save them to make a permanent channel.



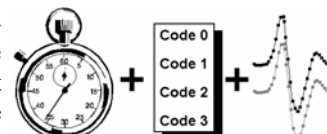
RealMark

A RealMark channel is exactly the same as a Marker, but with the additional ability to store a list of real numbers. RealMark channels can be created as the result of an active cursor measurement processor they can be created as a memory channel and saved to a permanent channel. An example of the use such a channel would be to store a list of blood gas parameters that are sampled from time to time during an experiment.



WaveMark

A WaveMark channel is exactly the same as a Marker, but also stores 1, 2 or 4 short waveforms (stored as 16-bit integers). These are normally used to store Spike shapes. The marker codes are usually used to store codes that represent the result of sorting the spike shapes into different classes. You can read more about sampling spike shapes in the *Spike sorting* chapter.



Channel lists

In many places where you are prompted for a channel, you can select a channel from a drop down list, or you can type a channel list. A channel list is a list of channel numbers or channel ranges separated by commas. A channel range is two channel numbers separated by two periods or a hyphen, for example 4..7, which is equivalent to channels 4, 5, 6 and 7. The channel range 7-4 is equivalent to channels 7, 6, 5 and 4. The following channel list:

```
1,3..5,7
```

means channels 1, 3, 4, 5 and 7. In most cases, Spike2 checks channel lists and removes channels that are not suitable for the operation. For example, if you open the `demo.smr` file supplied with Spike2, select an Interval histogram and type a channel list of 1..32 and then click on another field, Spike2 reformats the list to 2,3,31 as these are the only suitable channels. It is not an error for a channel list to include unsuitable channels, however it is an error for a channel list to include no suitable channels.

You can replace channel numbers with symbolic names using `m` for a memory channel or `v` for a virtual channel. For example `m1` for the first memory channel or `v2a` for the first duplicate of virtual channel 2. If you specify a range of duplicates of the same channel, the range expands by duplicates, not by channel numbers. For example: `2..2c` means 2, 2a, 2b, 2c. We allow up to 52 duplicates of a channel using upper case `A..z` for the second set of 26. So `2..2C` is the same as `2..2z, 2A..2C`.

Channel lists can also be used in script commands, for example: `ChanShow("1..4")`. Script commands that will accept this format describe the argument as `cSpC`. You can find more information about channel specifiers in the script language documentation.

Dialog expressions

Many dialogs in Spike2 accept an expression in place of a number. These expressions can be divided into two types: numeric expressions and view-based expressions.

Numeric expressions

A numeric expression is composed of numbers, the arithmetic operators `+`, `-`, `*` and `/`, the logical operators `<`, `<=`, `=`, `>=`, `>` and `?` and round brackets `(` and `)`. The result of a logical comparison is 1 if the result is true and 0 if the result is false. You may not have come across `?` which is used as:

```
expr1 ? expr2 : expr3
```

The symbols `expr1`, `expr2` and `expr3` stand for numerical expressions. The result is `expr2` if `expr1` evaluates to a non-zero value and `expr3` if `expr1` evaluates to zero. This can be used in the Cursor mode dialog to give a cursor position if a search fails based on some other information, for example:

```
Cursor(2)>Cursor(1) ? Cursor(2) : Cursor(1)
```

This evaluates to the position of the rightmost of cursors 1 and 2.

If you write expressions involving more than one operator, for example `1+2*3` you need to know if this is evaluated as `(1+2)*3` or as `1+(2*3)`. This is determined by the operator precedence level.

View-based expressions

These expressions follows the rules for numeric expressions and allow references to positions along the x axis. If a dialog field is documented as allowing expressions, and the field supplies an x axis position (for example a time), then you can use the following:

<code>Cursor(n)</code>	Where <code>n</code> is 0 to 9 returns the position of the cursor. If the cursor does not exist or the position is invalid, the expression evaluation fails.
<code>C0</code> to <code>C9</code>	This is shorthand for <code>Cursor(0)</code> to <code>Cursor(9)</code> .
<code>XLow()</code>	The left hand end of the visible x axis in seconds for a time view, bins for a result view, and x axis units for a XY view.
<code>XHigh()</code>	The right hand end of the visible x axis.
<code>MaxTime()</code>	The right hand end of the time axis in seconds for a time view and bins for a result view. It is not valid in an XY view.

`MaxTime(n)` The time of the last data item on channel `n` in a time view.

If a dialog field is documented as allowing expressions and the field supplies a y axis position, then you can use the following:

`HCursor(n)` Where `n` is 1 to 9 returns the position of the horizontal cursor. If the cursor does not exist or the position is invalid, the expression evaluation fails.

`H1` to `H9` This is shorthand for `HCursor(1)` to `HCursor(9)`.

`YLow()` The bottom end of the visible y axis.

`YHigh()` The top end of the visible y axis.

You can add `View(-1).` before these expressions to force the expression to be evaluated for the time view linked to the current view, for example `View(-1).Cursor(0)`. This is required when the current view is a result view and you wish to access timing information from the time view that the result view is based on.

Times as numbers

All times are in units of seconds. However, where a time is typed into a dialog you can use `{{{days:}hours:}minutes:}seconds` where the seconds may include a decimal point and items enclosed in curly brackets are optional. Each colon promotes the number to the left of the colon from seconds to minutes to hours to days. Times may only contain numbers and colons, white space is not allowed. One decimal point is allowed at the end of the time to introduce fractional seconds. We also allow a number with no colons to be followed by `ms` or `us` (with no spaces) to interpret the time in milliseconds or microseconds. So the following are equivalent: `1.6`, `00:00:01.6`, `1600ms`, `1600000us`.

Operator precedence

In the table, `LHS` means the value of the expression to the left of the operator as far as the next operator of same or lower precedence, `RHS` means the value of the expression on the right up to the next operator of the same or lower precedence. Where operators have the same level, evaluation is from left to right. The order from high to low is:

Level		Name	Return value
5	()	Brackets	Everything inside a pair of brackets is evaluated before considering the effect of an adjacent operator.
4	* /	Multiply Divide	LHS multiplied by RHS LHS divided by RHS. It is an error for RHS to be zero
3	+ -	Add Subtract	LHS plus RHS LHS minus RHS
2	< <= = >= >	Less than Less or equal Equal Greater or equal Greater than	If LHS less than RHS then 1 else 0 If LHS less than or equal to RHS then 1 else 0 If LHS equal to RHS then 1 else 0 If LHS greater than or equal to RHS then 1 else 0 If LHS greater than RHS then 1 else 0
1	?	Ternary operator	<code>LHS ? A : B</code> has the value <code>A</code> if <code>LHS</code> is not 0, and <code>B</code> if it is 0. Put spaces around the colon to distinguish it from a time.

`1+2*3` has the value 7 because multiply has a higher precedence level than add.

Script language compatibility

The expressions are compatible with the script language except for the use of `C0` to `C9` and `H1` to `H9` as shorthand for `Cursor(0)` to `Cursor(9)` and `HCursor(1)` to `HCursor(9)` and the use of colons, `ms` and `us` to denote times. If you use these in a script you will get syntax errors. However, you can use these constructs in strings passed as expressions to `CursorActive()` or `MeasureChan()`.

Copy current file path to clipboard

When working with file views (data, result, XY or text-based) you can copy the full path to the file (if the view has an associated file) by right clicking on the view title bar and then selecting the **Copy path** item that appears. This is especially useful if you have set the operating system to not display full path names. This is an experimental feature, added at version 8.02.

Data view keyboard shortcuts

The following shortcut key combinations can be used in a time, result view or XY view except for **Ctrl+Shift+Left/Right** and **Alt+Left/Right**, which are only available in a time view. Items marked *Always available* can be used, regardless of the current view.

Key	Operation
Left arrow	Scroll 1 pixel left.
Right arrow	Scroll 1 pixel right.
Shift+Left	Scroll several pixels left.
Shift+Right	Scroll several pixels right.
Ctrl+Left	Scroll half a screen left.
Ctrl+Right	Scroll half a screen right.
Ctrl+Shift+Left/Right	Time views only. If cursor 0 is active, search for the next/previous feature and scroll the screen to make it visible.
Alt+Left Alt+Right	Time views only. Search selected event channels for the previous/next event that is nearest to the screen centre and make it the screen centre or make a sound if there are no selected channels or no more events.
Alt+Shift+Left/Right	Time views only. Jump to the next display trigger point (only if the display trigger is enabled).
Home/End	Scroll to the start/end of the data.
Ctrl+n	Where n is 0 to 9. Fetch vertical cursor 0 to 9. If the cursor does not exist it is created. Cursor 0 exists only in time views.
Ctrl+Shift+n	Where n is 0 to 9. Centre the window on the cursor, if it exists. Beware that Microsoft have grabbed Ctrl+Shift+0 for IME language from Vista onwards. You can get instructions to defeat this in the <i>Technical support</i> , Frequently asked questions.
Ctrl+Shift+A	Fetch all vertical cursors.
Alt+n	Where n is 1 to 9. Fetch horizontal cursor n to the first visible channel with a Y axis.
Ctrl+A	Select all channels. If all channels are selected, unselect all channels.
Ctrl+Alt+A	Abort sampling; unless no data has been captured or you have been running for only a very short time you will be asked to confirm this.
Ctrl+C	Copy the image of the view to the clipboard.
Ctrl+E	Expand (zoom out) the data view around the left edge of the window.
Ctrl+I	Reduce (zoom In) the data view around the centre of the window.
Ctrl+Shift+I	Open the View Information dialog for the current view (where supported).
Ctrl+K	Show the x axis dialog.
Ctrl+N	Open a new data file choice dialog. Always available.
Ctrl+O	Open the file open dialog. Always available.
Ctrl+P	Print the current view. Always available.
Ctrl+Q	Optimise selected channels in the current view. If none selected, optimise all channels.
Ctrl+L	Open the Evaluate window to run script commands. Available unless a script is running.
Ctrl+R	Reduce (zoom in) the current view around the left edge of the window.
Ctrl+Shift+R	Reset sampling. You will be queried unless no data has been captured or you have been running for a very short time. Always available when sampling.
Ctrl+S	Save the current view (where supported). Equivalent to the File menu Save command.

Ctrl+Alt+S	Start sampling (if a data view is ready to sample) or stop sampling if already sampling data.
Ctrl+T	Create a TextMark during sampling (if the TextMark channel exists).
Ctrl+U	Expand (zoom out/Up) the current view around the centre of the window.
Ctrl+Shift+W	Toggle writing data to disk on all channels. Always available when sampling.
Ctrl+Y	Show the y axis dialog.
Ctrl+Z	Undo the last undoable operation for the current view.
Ctrl+Break	Break out of long drawing or calculation operations.

Text view keyboard shortcuts

Text views have more keyboard short cuts than any other area of Spike2. We have grouped them by function to make the huge list more digestible.

Find Replace and Bookmarks

The Find and Replace commands can be accessed from the Edit menu, from the Edit Toolbar and by keyboard short cuts:

Key	Operation
Ctrl+F	Open the Edit menu Find dialog. In addition to searching for text you can also use this dialog to bookmark all matching text.
Ctrl+H	This short-cut key opens the Edit menu Replace dialog.
F3	Repeat the last find operation in the same direction. You can use the toolbar to search forwards or backwards.
F2	Move the text caret to the next bookmark. You can use the edit toolbar to move to the next or previous bookmark.
Ctrl+F2	Toggle bookmark on the current line. You can use the edit toolbar to set or clear a bookmark and to clear all bookmarks.

Bookmarks tag a line for future reference. They are displayed as a blue mark to the left of the text. Bookmarks are kept as long as the current file is open; they are lost when you close the file. The easiest way to use a bookmark is from the Edit Toolbar. You can show and hide this from the Edit menu (when a text-based window is active), or by clicking the right mouse button on any toolbar or on the Spike2 application title bar and using the pop-up context menu that appears.

Text caret control

The text caret is a flashing vertical bar that indicates the current position. Do not confuse this with the I-beam mouse pointer which does not flash and which indicates the mouse position. Each time you click and release the left mouse button (we assume you haven't swapped the mouse buttons), the caret moves to the nearest character position to the click point. To select text with the mouse, click at one end of the text you want to select and drag (move the mouse with the button held down) to the other end of the text. You can also use the keyboard to move the caret and select text:

Key	Operation (+Shift to extend a text selection, +Alt+Shift for rectangular)
Left arrow	Move the caret one character left. At the start of a line it wraps to the previous line end.
Right arrow	Move the text caret one character right. You can move it into uncharted territory beyond the end of the line. It does not wrap to the next line.
Up arrow	Move up one line.
Down arrow	Move down one line.
Ctrl+Left Ctrl+Right	Move one word to the left/right. Words are defined to be useful when operating on scripts.
Ctrl+Up/Down	Scroll the text window up and down, leaving the selection unchanged.
Alt+Up/Down	Move the lines containing the selection up or down by one line.

End	Move the caret to the right of the last character on the line.
Home	Move the text caret to the first non-white space character in the line. If already at that point, move to the start of the line.
Alt+Home	Move the text caret to the start of the current line.
Ctrl+End	Move the text caret to the right of the last character in the file.
Ctrl+Home	Move the text caret to the left of the first character in the file.
Ctrl+] ([)	Start of next (previous) paragraph (after empty line)
Ctrl+\ (\/)	Word part right (left).
PgUp	Move upwards by one window of lines.
PgDn	Move downwards by one page of lines.
Insert	Swap between insert mode caret and over-type caret _

Cut Copy Paste Delete Undo and Redo

Some of these operations are also available from the Edit menu and the main toolbar.

Key	Operation
Ctrl+A	Select all the text in the document.
Ctrl+C	Copy selected text to the clipboard. If no text is selected, nothing is copied. Some keyboards have Ins in place of Insert.
Ctrl+Insert	
Ctrl+Shift+T	Copy the current line to the clipboard.
Ctrl+V	Paste the contents of the clipboard into the text at the caret. If there is a selection, the selection is replaced.
Shift+Insert	
Ctrl+X	Cut the selected text and copy it to the clipboard.
Shift+Del	Cut the selected text and copy it to the clipboard.
BackSpace	Delete the selection or the character to the left of the text caret.
Del	Delete the selection or the character to the right of the text caret.
Ctrl+Del	Delete word right. Add Shift to delete to the end of the line.
Ctrl+D	Duplicate the selection. If it is empty, duplicate the line containing the text caret.
Ctrl+Shift+L	Delete the current line.
Alt+Up/Down	Move all the lines in the current selection up or down by one line.
Ctrl+Z	Undo the last interactive text operation. The editor supports more or less unlimited levels of
Alt+Backspace	Undo.
Shift+Ctrl+Z	Redo the immediately previous Undo operation.

Miscellaneous

These commands do not fit into any other category!

Key	Operation
Ctrl+U	Convert the selection to upper case. Add Shift for lower case
Ctrl+Add, Sub	Change font size (Add and Sub are numeric keypad + and -). You can also change the displayed text size by holding down the Ctrl key and using the scroll wheel on your mouse (if you have one). The View menu Standard Display command will remove any zooming. The script equivalent is ViewZoom().

Indent and Outdent

The structure of Spike2 scripts is often made clearer by indenting program structures. To make this easier, you can indent and outdent selected blocks of text to the next or previous tab stops. The tab size is set in the Edit menu Preferences option.

Key	Operation
Tab	If there is a multi-line selection, all lines included in the selection are indented so that the first non-white space character is at the next tab stop. If there is no selection, a tab character is inserted (or spaces to the next tab stop depending on the Edit menu Preferences settings).
Shift+Tab	If there is a multi-line selection, all selected lines are out-dented so that the first non-white space character on the line is at the previous tab stop. If there is no selection, the text caret moves to the previous tab stop unless it is already at one.

Text view features

The Log view, script views, the output sequencer and text windows created by a script all count as Text views and they have common features. This section deals with editing features: Drag and drop, Virtual space, Multiple selections, Rectangular selections and Read only files.

Drag and drop

The editor supports drag and drop of text both within Spike2 and between Spike2 and other applications that support it (for example the Spike2 Help system). Spike2 also supports drag and drop for rectangular text areas.

Operation	Method
Move block	Select the text to move. Move the mouse pointer over the selected text and hold down the left mouse button and drag. The mouse pointer will indicate that you can now drag the text and the text caret will show the insertion point. Drag the text to the desired insertion point and release.
Copy block	Select the text to copy. Hold down the <code>Ctrl</code> key and move the mouse pointer over the selected text, click and drag. A small + symbol indicates the copy operation and the text caret will show the insertion point for the duplicate. Drag the text to the target position and release the mouse button to duplicate the text. The <code>Ctrl</code> key must be down when you release the mouse button or the operation will move the text.

Virtual space

If virtual space is enabled you can position the caret beyond the end of the text in a line by clicking in a blank area with the mouse, or using the cursor right key. You cannot position the text caret below the last line of text. When the caret is beyond the end of the line, it is said to be in *virtual* space. If you type with the caret in virtual space, space characters will be added to fill in the virtual space up to the text you type.

There are two main uses for virtual space: to add comments without having to space along to the required column, to make rectangular selections without having strange visual effects due to short lines.

If you use the script language to manipulate the text caret and make selections, virtual space is ignored; a caret in virtual space will be treated as if it is at the end of the line. There are no script commands that will move the caret into virtual space. If there is a requirement for the script to report or use virtual space, we will extend the script in a compatible way to incorporate it.

Multiple selections

You can make multiple selections in a text view. To do this, hold down the `Ctrl` key and click and drag. Each time you make a new selection in this way it becomes the current selection; all previous selections as shown with a different selection colour. When you have a multiple selection, each selected area has a flashing text caret at the insertion point. If you type characters, these will appear at all insertion points, replacing all the selected text. If you use the delete or backspace key, all selected text will vanish. Note that pasting into a multiple selection will clear all the selected text, then insert the pasted text at the current (last made) selection.

Multiple selection can be useful when you want to move several non-consecutive script functions to make

them consecutive. Select all the sections you want to move, use `Ctrl+X` to cut them all, release `Ctrl`, click at the insertion point, then use `Ctrl+V` to paste the result.

Select rectangular text area

You can select, cut, paste and drag rectangular selections within Spike2. To select a rectangular area hold down the `Alt` key then select text with the mouse. The point where you hold down the mouse button will be one corner of the selection, the point where you release the mouse will be the other corner. You can use this feature to change the alignment of comments in a script, or to convert a single column of numbers into multiple columns. You can also paste such text into other applications as plain text.

A rectangular selection will paste within Spike2 as a rectangular selection. Beware that `Ctrl+X` (cut) on a rectangular selection followed by `Ctrl+V` (paste) will not leave the text unchanged (unless you select the text from bottom to top). The cut operation will leave a vertical flashing line (assuming a fixed pitch font) with the insertion point marked by a more visually obvious caret. The paste operation will paste the cut text as a rectangular block at the insertion point. A rectangular selection is a multiple selection.

Read only text

If you open an output sequence or script file that is held in a read only file, the text window will also be marked read only and you will not be allowed to change it. This is to allow users to protect sequences and script from inadvertent change. If you want to edit such a text file you have two options:

- 1) Close the file, remove the read only status on disk, then open it
- 2) Save the file to a new file with a different name

You can use the `Modified()` script command to get and change the read only status of a text file. However, if the original disk file is marked read only, using `Modified()` to allow you to edit the text will not change the status of the disk file, so you will not be able to save it to the same file name.

The Spike2 command line

When Spike2 starts, it checks the command line for option switches and files to load. If there is no command line, Spike2 looks in the current directory for `startup.s2s` and runs it if it exists. If `startup.s2s` is used or the command line loads a file, start up messages that need a user response are suppressed.

The command line holds options and file names separated by white space characters (space and tab). If a file name contains spaces, you must surround the file name with quotation marks. Options start with `/` or `-` followed by a character to identify the option.

`/M` When Spike2 starts, it checks if it is already running and quits if it is. `/M` removes the check, allowing multiple copies to run. You need a Spike2 licence for each copy except when using multiple synchronised 1401s to capture related data on one computer under the control of a single operator, when one licence is sufficient. Set separate file names for each 1401 in the Automation tab of the Sampling Configuration.

`/Un n` is 1-8 to select a 1401 when you have more than 1. The default is `/U0`, which uses the lowest-numbered unused 1401. You set a device number in the CED 1401 device settings in the Device Manager (My Computer->Properties->Hardware).

`/Q` Quiet startup. Suppress all message boxes and the Spike2 "splash screen".

The remaining items in the command line are assumed to be file names. Spike2 attempts to load the files in command line order (from left to right). The files must have extensions so that the file type is known. If a script file is included in the command line, Spike2 runs it before continuing with the remainder of the command line.

As an example, suppose we want to launch Spike2 so that it automatically opens a data file called `lots of data.smr` and runs `doit.s2s` to process it. Follow these steps:

1. Create a short cut to `sonview.exe` (this is the Spike2 program).
2. Right-click on the new short cut and select **Properties** and open the **Shortcut** tab.
3. Add "`lots of data.smr`" `doit.s2s` to the end of the **Target** field.
4. Set the **Start in** field to the folder that contains your files.

5. Click OK.

This example assumes that both files are in the same folder. You could also have included the full path to each file in the command line.

Shell extensions

Previous Spike2 versions added shell extensions that displayed additional information in Windows Explorer when the mouse pointer hovered over Spike2 data and script files. In Windows NT2000 and XP you could also display file comments. These shell extensions were held in `SonCols.dll` and `SonInfo.dll` in the Spike2 installation folder. Regretfully, we have removed these shell extensions from version 8, because:

1. From Vista onwards, Microsoft has removed the underlying operating system support used by `SonCols` (which displayed file comments in Explorer if you enabled the Comments column).
2. There are problems with library versions that make installing `SonInfo` problematic. These can prevent libraries being updated correctly when new versions of Spike2 are installed, leading to obscure crashes.

We may reinstate them at a future date if we can do it without causing problems.

Removing the shell extensions

To remove shell extensions installed by a previous version of Spike2 without uninstalling the old copy of Spike2, open a command prompt and navigate to the old Spike2 installation folder and type:

```
C:\>cd \Spike7 Change to the Spike2 folder
C:\Spike7>regsvr32 /u SonInfo For all Windows versions
C:\Spike7>regsvr32 /u SonCols Only for NT2000 or XP
```

You may need to run as administrator to do this (particularly for `SonCols`). To do it:

1. Click on the start menu at the bottom left of the screen.
2. In the box with "Search programs and files" type `cmd`
3. A list of matching files should appear including `cmd.exe`.
4. Right-click on `cmd.exe` and select "Run as administrator".
5. In the command prompt you can then issue the commands listed above to remove the shell extensions.

64-bit operating systems

Before Spike2 version 8, all versions of Spike2 were 32-bit programs, even if the operating system was 64-bit. From Spike2 version 8, by default, we will install a 64-bit version of the program on 64-bit operating systems and a 32-bit version on 32-bit operating systems. The advantages of 64-bit code on a 64-bit operating system are:

1. The 64-bit version of the CPU has more registers and instructions; "typical code" runs maybe 10% faster.
2. Calls into the operating system do not transition from 32-bit Spike2 code to 64-bit system code and back.
3. A 32-bit program is limited to a 4 GB memory space, and of this, quite a bit (typically 1GB) is reserved for the 32-bit operating system. With a 64-bit program you can use a lot more memory.
4. The new `son` library uses 64-bit integers to hold times; these are much faster to manipulate with 64-bit registers than with 32-bit registers.

There are some disadvantages, too:

1. 64-bit code is typically larger (for example the `sonview.exe` file is 20% larger as 64-bit code).
2. Pointers in memory are double the size, so memory usage increases.

Unless your system has limited installed memory, we recommend that you use the 64-bit version of Spike2 on a 64-bit operating system.

Mouse buttons

Throughout the Help we assume that you have not swapped your mouse buttons over. That is, the left button is the standard click and the right button is used for context menus. If you have swapped your mouse buttons, then you must interpret the descriptions accordingly.

Program files installation

This section applies to both Spike2 version 7 and 8, so we refer to installation directories as `SpikeN` where `N` stands for 7 or 8 as appropriate.

Previously, we installed Spike2 in a top level folder such as `C:\SpikeN`. While this practice was satisfactory when used with earlier versions of Windows, it has become deprecated, mainly for security reasons. We have altered Spike2 so that it can be installed in the protected `C:\Program Files` folder as expected by Microsoft. Some users will prefer to keep their existing arrangements so we still support installation into `C:\SpikeN` or similar. This support will be removed at some future point.

The main change caused by installing inside `C:\Program Files` is that this location is expected to be read-only as seen by users. While there is no problem in storing data files, sampling configurations, scripts and other files in `C:\SpikeN`, you are not allowed to modify anything inside `C:\Program Files`. Therefore as part of the Spike2 installation new folders are created for the current user's data, for data shared between all users and for data generated by the application.

New folders created by the installer

User data folders

User data folders are used to hold data that is directly saved or loaded by the user, for example data files, sampling configurations and scripts. Two such folder are created by the Spike2 installer:

- Current user data: called `SpikeN` in the `My Documents` folder for the current user. When installing Spike2 inside `C:\Program Files`, the installer puts all data intended for the user's direct use here. The folder contains the `Data`, `Scripts`, `Sequence`, `ExtraDoc` and `TrainDay` folders and an empty `Include` folder for your own script include files. If there are other Spike2 users on the system, they should create a `SpikeN` folder inside their own `My Documents` folder. When searching for a user data folder, Spike2 seeks the current user data folder first and if it is not found, looks for the all-users data folder:
- All-users data: called `SpikeNShared` in the `Documents` folder provided for all users of the system. The folder contains an empty `Include` directory for shared script include files. This folder is available for multiple system users to store shared data.

Application data folders

Application data folders are used to store data that is generated by the application without any direct user involvement, for example the default filter bank settings and the last-used and default sampling configuration. As for the user data folders, two application data folders are created by the Spike2 installer:

- Current-user specific. This folder location varies with the operating system version but in Windows7 the folder is `C:\Users\User\AppData\Local\CED\SpikeN` where `User` is the user name. If a different user is logged-on to the system they should either create their own matching directory or rely upon the shared all-users application data directory. When searching for an application data directory, Spike2 generally looks for the current user's application data directory first and if this is not found looks for the all-users application data directory:
- All-users application data. This folder location varies with the operating system version but in Windows7 the folder is `C:\ProgramData\CED\SpikeN`. If there are multiple Spike2 users on the system this directory ensures that there is always a suitable folder available for application data.

Moving your files to a new-style location

After installing inside `C:\Program Files`, the original `C:\SpikeN` folder (or whatever was used) will be unchanged, holding both your own files and the previous installation of Spike2. You can continue to load

sampling configurations from and save data files into `C:\SpikeN` just as before and some users may prefer to do this, perhaps after cleaning up (see below). Alternatively you can choose to keep your data and other files in the new location. To do this, copy (or move) all `.smr`, `.sxy`, `.s2r`, `.s2rx`, `.s2c`, `.s2cx`, `.s2s` and `.pls` files, plus any other files you might want to keep and any folders of your own into the `SpikeN` directory inside your `My Documents` folder. Once you do that you will be able to use your data files, scripts, sampling configurations and other files from the new location.

There can be issues in moving files. If you copy (rather than move) files to the new location and if they are referred to in scripts or sampling configurations by their full path name, the original versions of the files will be used, which can cause confusion. After moving your data to a new location you should check the following:

Application-level issues:

- The Sampling page of the Preferences dialog sets the folder for the temporary `smr` data file that is created by sampling. By default in older versions of Spike2 this was set to be the application data folder, so you may find that it is set to `C:\SpikeN`. You can set this as you prefer, or clear the path and click OK and Spike2 will set it to an application data folder. Do not set a folder in `C:\Program Files` as this may cause problems with write permissions.
- Global resources - some options for the global resource file location use the Spike2 installation folder. While this location will still be used, Spike2 now uses a list of folders which are searched in turn for the required file. The list starts with the current user's data folder inside `My Documents` - if you are using global resources you should make sure the resource files are in a suitable location; the `SpikeN` folder inside `My Documents` is probably the best place.
- The Sample and Script bars hold lists of sampling configuration and script file names. These lists hold the complete path to the file so if you use either of these you should update the file names so that they point at the new file locations.

Data and XY file issues: There should be no problems with the data files themselves as they do not contain any information relating to folder locations. However if a background image file has been set up for a data or XY file the background image settings will contain a complete path to the image file which should be updated.

Sampling configurations: Sampling configuration files can contain a number of folder paths which may need to be adjusted. Note that (as documented here) Spike2 now only writes sampling configuration files using the new XML file format with a new `.s2cx` file extension so your adjusted sampling configuration files will have to be converted to the new format by using them for sampling before they can be saved as XML. Therefore the changed files will all use the new file name extension and your old configuration files will remain unchanged. You should take care to only use the new-style configuration files once they have been adjusted and checked.

- Sequence file. If the sampling configuration Sequencer page specifies a sequence file, you may need to edit the file path to point to the new file location.
- Automatic file-naming. If the sampling configuration Automation page sets a folder for automatically-named files this path might be unsuitable for the new arrangement and should be adjusted.

Sequence files: If the file uses `#include` commands you may need to check that these still work.

Script files: If the file uses `#include` commands you may need to check that these still work. If the script uses file path string constants, these may need adjusting. Any use of `FilePath$(2)` to get the Spike2 installation folder is likely to give problems as you should not create or write to files inside `Program Files`. The parameters to `FilePath()` have been adjusted to allow you to find the `My Documents` folder where Spike2 data files are installed - we suggest that you change to using this location wherever possible.

Cleaning up the old installation location without moving your own files

If you change from `C:\SpikeN` to `C:\Program Files` and you want to leave your own files in `C:\SpikeN`, you may want to clean out redundant files. If the new installation is a different Spike2 major version, for example moving from version 7 to version 8, the easiest way to remove the old installation without deleting files that you created is to uninstall the old version using the system control panel. However, if you are moving within the same major version this would delete the current version; you can do the following instead:

- Delete the `1401`, `BaseDemo`, `ExtraDoc`, `Export` and `import` folders from the old install location. If you not put any of your own files into them, you can also remove the `data`, `scripts`, `sequence`, `include` and `trainday` folders.
- Inside the old Spike2 installation folder itself, delete all `.exe`, `.dll`, `.chm`, `.s2l`, `.t14` and `.tip` files. These are files specific to the Spike2 installation and will have been replaced by the newly installed files.

4: Sampling data

Sampling data

If you worked through the *Getting started with Spike2* section you already have most of the skills to sample data. Sampling a new document is the same as working with an old one, except that the length increases.

Sampling configuration

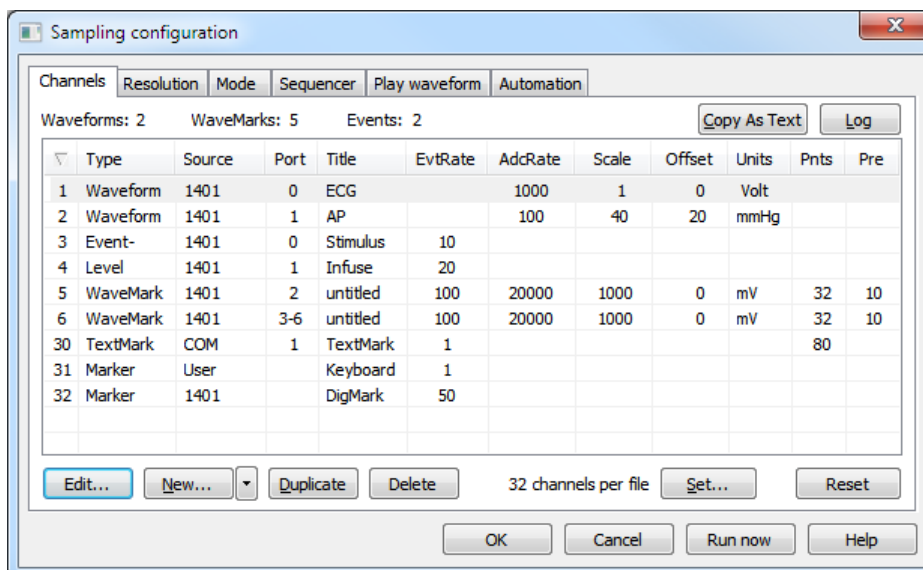
Before you start to sample data you must set a sampling configuration. This defines the channels to capture, the file type and timing resolution, how the sampling is to start, time limits for sampling and outputs to generate during sampling. This is done in the **Sample** menu **Sampling Configuration** dialog. This dialog holds several pages activated by the tabs at the top, each page controlling a different aspect of data capture.

The title bar holds the name of the configuration file from which the configuration was read. There is a * at the end of the name if the configuration has been changed. The **OK** button accepts the current state, **Cancel** rejects any changes you have made since you opened the dialog and the **Run now** button opens a new data document, ready to sample. You can also open this dialog from the toolbar.

The sampling configuration also holds windows positions, channel arrangements and settings for on-line data processing into result and XY views or into time view data channels. These properties are set by opening a file for sampling, setting the window, channel and processing as you need it, then saving the configuration. To see the script language equivalent of a configuration, load or set the desired configuration, turn recording on, open a new file ready to record, then turn recording off.

Channels

The **Channels** tab lists the channels to sample. Channels are taken from a 1401 interface, from the keyboard (TextMark data can also be read from a serial port), or from a Talker. The left-hand column is the channel number; the remaining columns list the channel type, the physical port and the title, then data that varies with the channel type. You can click the header of the channel list to sort the channels by most of the fields.



The top line displays counts of Waveform channels, ports used for WaveMark channels and event channels plus controls to copy the configuration as text to the clipboard or to the Log window. You edit the channel settings by double-clicking on a channel, or by selecting a channel with the mouse or keyboard and clicking the **Edit** button. The **Reset** button deletes all editable channels and sets a standard sampling state.

Duplicate

The **Duplicate** button is enabled if it is possible to duplicate the selected channel. In all cases, there must be spare channels in the file. You can duplicate a 1401-based channel if there is a spare port for that channel type. You can duplicate a Talker-based channel if the next higher item in the Talker has the same channel type.

Copy As Text, Log

These two buttons render the sampling configuration as text in a format suitable for a laboratory notebook. **Copy As Text** places the text on the clipboard. **Log** copies the text to the Log window (so you can print it). The `SampleConfig$()` script command gets this text with additional formatting opportunities.

Set channels

The **Set...** button opens a dialog in which you can set the maximum number of channels that can be stored in a data file created by the File menu **New** command.

Reset

The **Reset** button clears all the sampling information in the Sampling Configuration dialog to a default state. As this is destructive, you are asked if you really want to do it. Even if you do **Reset** your configuration, changes made can be cancelled with the **Cancel** button. The **Reset** button does not clear information on duplicate windows and channels; see the Sample menu->Clear configuration command for this.

Set maximum channels

The **Set...** button in the Sampling configuration dialog and the **Channels...** button in the Export As dialog both open the **Set channels in new data files** dialog. The minimum number of channels in a data file is 32 and the maximum is currently limited to 400 (64-bit files have a theoretical limit of 65535 channels, but this is not yet supported by Spike2).


In the Sampling configuration dialog, if you reduce the number of data channels and there were channels defined with numbers above the new limit, these higher numbered channels are deleted from the configuration.

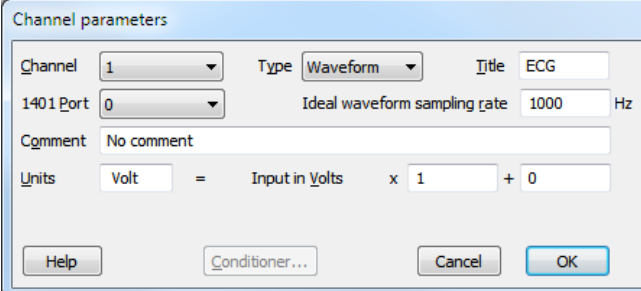
Spike2 data file sampling will allow you to sample up to 100 channels (you will need a 1401 interface with expansion top boxes to use all of these channels). However, you may wish to use more channels than this for derived data generated by a script or by interactive analysis.

Setting a larger number of channels increase the size of the file header, which holds the channel title and any units and sample rate information. It does not allocate data space in the file, so apart from a small increase in the channel header size per channel and in memory usage per file, there is no other penalty. The 32-bit filing system does not allow you to change the number of channels once the file is created. The 64-bit filing system theoretically can add more channels after the file has been created and used, but there is no support for this in Spike2 yet. It is a good idea to allocate enough channels for your foreseeable data analysis needs. You can, of course, export a file to a new file with more channels.

If you will share your files with people using old versions of Spike2 you should be aware that before Spike2 version 4.02, only (32-bit) data files with 32 channels were readable. Version 5 reads 32-bit files with up to 256 channels (5.15 onwards can read 400 channels).

Create a new channel

 The **New...** button creates a new channel at the lowest free channel number and opens the **Channel parameters** dialog where you can edit the new channel to the desired state and set the channel number. To make several similar channels, set up the first channel, then click **Duplicate** to insert a new channel at the next free channel and physical 1401 port with the same settings. To remove an unwanted channel, select it and click **Delete**.



You can also use the small button to the right of **New...** to select the type of the new channel from a drop-down list that also includes the names of any Talker devices that can be used as data sources.

When you click the **Edit...** button or double-click on a channel, a new dialog opens where you set the channel characteristics. The first five fields apply to all data types.

- Channel The channel number identifies the channel. Channels 30 to 32 are reserved for Text, Digital and keyboard markers. The remaining channels are for waveform, WaveMark and event data and data from Talkers. You can change a channel number, but not to a channel that is already in use.
 - Type The type of the data to sample on the channel. You can set any of the following:
 - Off The channel is unused, equivalent to deleting the channel
 - Waveform The channel holds waveform data
 - Event- Event channel timed on the falling edge of the data
 - Event+ Event channel timed on the rising edge of the data
 - Level Event channel with the times of both data edges saved
 - TextMark The channel holds (short) text messages and their times.
 - Marker The data is an event with either a keyboard character or a digital value attached
 - WaveMark The data is a small waveform fragment, usually a spike shape
 - Title Up to 8 characters to identify the channel.
 - 1401 port For a waveform or WaveMark channel, this is the 1401 ADC input number. For an event channel, this is the digital input port number. Marker channels do not use this field. You will not be allowed to sample if you request a physical port number that does not exist in your 1401.
 - Comment Some text to give more information about this channel.
- The Conditioner... button is present for Waveform and WaveMark data types and opens the signal conditioner dialog if a conditioner exists for the channel.

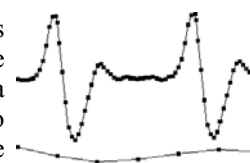
Maximum sampling rates

The ultimate limit on the sampling rate is how fast data can be transferred from your 1401 to the host. If you use a USB 2 interface, this can manage rates between 10 and 40 MB/second (depending on the type of 1401), so this is not usually a limit. However, if you have an early Power1401 with a USB 1 interface or if you plug your USB 2 capable 1401 into a USB 1 port, you will be limited to around 1 MB/second. The PCI interface has a similar limit to USB 1. If your 1401 has a choice of interface, USB 2 is by far the quicker.

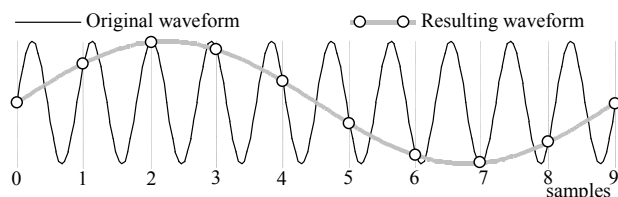
Another limit on throughput is the maximum ADC sampling rate that your 1401 can achieve. You should consult the Owners Guide that came with your 1401 to get the details. The sampling configuration allows you to specify the target 1401 type. You are limited to achievable rates for the selected target.

Waveform channels

The waveforms you record are continuously changing voltages. Spike2 stores waveforms as a list of 16-bit numbers in the range -32768 to 32767 that represent the waveform amplitude at equally spaced time intervals. The process of converting a waveform into a number at a particular time is called *sampling*. The time between two samples is the *sample interval* and the reciprocal of this is the *sample rate*, which is the number of samples per second. The dots in the diagram represent samples; the lines show the original waveform.



Minimum sample rate



The waveform sample rate must be high enough to represent the data. The sample rate must be at least double the highest frequency contained in the data. If you do not sample fast enough, high frequency signals are aliased to lower frequencies (as shown as the resulting waveform in the diagram). On the other hand, you want to sample at the lowest frequency possible, otherwise your disk will soon fill. Unlike many data capture

systems, Spike2 lets you capture different waveform channels at different rates to minimise data file sizes.

Use of filters

Many users pass waveform data through amplifiers or signal conditioners with filter options such as the CED 1902 to limit the frequency range. Some transducers have a limited frequency response and require no filtering.

Input connections

Channel	8	9	10	11	12	13	14	15	Gnd
Power (37 pin)	28	29	30	31	32	33	34	35	1-19
Power (44 pin)	33	34	35	36	37	38	39	40	3-10

Connect your waveform channels to the 1401 ADC inputs. Channels 0-7 (0-3 for a Micro1401) are the labelled BNC connectors. Channels 8-15 are on the Power1401 rear panel 37-way Cannon connector except for late model mk I and all Mk II, which have a 44-way high density connector and each signal has its own ground. Pin numbers are given in the table. If you have an ADC expansion fitted you will have more channels; see the accompanying documentation for the connections. Power1401 top boxes can reassign the rear panel channels. The standard input Voltage range is ± 5 Volts. If you have a ± 10 Volt system check that the Voltage range is set correctly in the Edit menu Preferences.

Waveform dialog

The Waveform channel dialog has all the standard fields, plus:

Ideal rate Set the **Ideal waveform sampling rate** field to the desired sampling rate for this channel. The actual sample rate will be as close to this ideal rate as possible. The **AdcRate** column in the Sampling Configuration dialog displays the actual rate (in red if it is more than 20% different from the desired rate). If the rates differ too much, adjust the optimisation and clock settings in the Resolution tab.

Units This field holds the waveform units, up to 5 characters long. The following fields are the *scale* and *offset* to convert the input into user units.

$$\text{input in user units} = (16\text{-bit input value}/6553.6) * \text{scale} + \text{offset}$$

This is arranged so that with a ± 5 Volt input range, the *scale* is the number of user units for every one Volt increase in input; *offset* is the value represented by 0 Volts at the input. In this case:

$$\text{input in user units} = \text{Volts at the input} * \text{scale} + \text{offset}$$

However, some systems have ± 10 Volt inputs. In this case:

$$\text{input in user units} = 0.5 * (\text{Volts at the input}) * \text{scale} + \text{offset}$$

As an example, consider a situation where a waveform represents a position. 1 Volt is equivalent to 10 mm and 3 Volts is equivalent to 50 mm and the 1401 has ± 5 Volt input range. In this case you would set:

$$\text{scale} = (50 - 10) / (3 - 1) = 20.0 \text{ mm V}^{-1}$$

$$\text{offset} = 10 - (1 \text{ Volt}) * \text{scale} = -10.0 \text{ mm}$$

$$\text{Units} = \text{mm}$$

To display in metres in place of mm, set *scale* to 0.02, *offset* to -0.01 and *units* to m. You can calibration a channel after sampling from known input data with the Analysis menu **Calibrate** command.

A common requirement is to display the sampled data in millivolts. In this case, just set the Units to mV, the scale to 1000 and the offset to 0.

For the scaling to work as expected, the Edit menu Preferences option for Voltage range (5 Volt range or 10 Volt range) must be set correctly for your 1401; this should happen automatically for a modern 1401 where we can read back the voltage range. Since version 6.12, we store the voltage range in the sampling configuration and adjust matters as appropriate. If you open an old sampling configuration, we will assume that the voltage range is whatever is currently set.

Note that the scale value associated with a channel of sampled data and accessed by the `ChanScale()` script command is the same as the scale value defined here for 5 Volt systems, but will be double the value here for 10 Volt systems. The scale and offset values defined here are also set by the `ChanCalibrate()` script command.

Event data

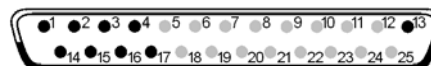
Spike2 stores time stamps efficiently as integer multiples of the time resolution set in the Resolution tab of the sampling configuration. Simple time stamps with no other attached data are called *events*. Each event uses 8 bytes of storage. The 1401 recognises events as changes of state of TTL compatible signals connected to the 1401 digital input bits 15-8. We refer to these inputs as *event ports* 7-0. There are three types of event:



- Event-** Spike2 saves the time of the falling edge of the input signal. The minimum input pulse width is 1 μ s; wider is better.
- Event+** The same as **Event-**, but Spike2 saves the time of the rising edge.
- Level** Spike2 saves the time of both edges. Pulses should be a minimum of 20 μ s wide or the time resolution set for sampling, whichever is the larger. Do not use this type unless you need the times of both edges. From Spike2 version 8.00, Level events are actually a form of Marker data and the state of the event (low going or high-going) is stored in the marker code.

Micro1401 and Power1401 event port connections

Event ports 0 and 1 are BNC sockets on the front panel. If you have the optional Spike2 top box, event ports 7-2 are also on BNC sockets, otherwise you must use digital input bits 15-10. The digital input connector is on the rear panel. If you want to use the rear panel digital input connector for event ports 0 and 1 there is an option in the Edit menu Preferences... to use the rear panel connector for all events.



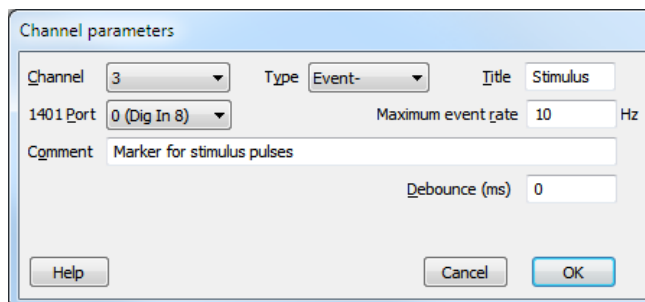
Input connections (digital input)

Digital input bit	15	14	13	12	11	10	9	8	Gnd
Digital input pin	1	14	2	15	3	16	4	17	13
Event port	7	6	5	4	3	2	1	0	

The pin connections for the digital input connector are the same for all 1401s:

Event dialog

The event channel dialog is similar to the waveform dialog. There is no Units field and there are new fields for the Debounce period and the Maximum event rate. If two events are closer together than the debounce period, the second event is ignored. You can also set a negative value (-1 recommended) to convert simultaneous events into events at consecutive clock ticks. Set this field to 0 to disable this feature; setting 0 saves processing time.



The Maximum event rate should be set to your estimate of the maximum mean event rate sustained over a few seconds. This is not the maximum instantaneous rate, which may be much higher. Spike2 uses this information to optimise buffer space allocation.

As an example, an event channel might have a mean rate of 30 events per second, but it could have an instantaneous maximum rate of 1 kHz if two events fell within 1 millisecond of each other. In this case, the rate you should enter is 30 Hz, not 1000 Hz.

Debounce

This field appears for Event-, Event+ and Digital Marker channels. It sets the minimum acceptable interval between consecutive events in milliseconds. Events closer than this to the previous event are not saved to disk. This is typically used when events are logged from a mechanical switch. Mechanical switches commonly have bouncy contacts, such that when you activate the switch you can get multiple switch actions in a short period of time. You can use this setting to keep the first of a series of pulses and reject all the extra ones within the debounce period of the first. There is a time penalty for setting this value non-zero, so you should only use this option if you need it.

In terms of system performance it is always better to fix the bouncy signal (for example by using a better switch or an electronic switch debouncer) as all the extra events use up 1401 resources to capture them and transfer them to the host computer where additional resources are used up in eliminating them.

Level event channels

There is no debounce option for Level event channels. If you sample to a 32-bit smr file we save all the data, as sampled; duplicated event times may cause some problems with subsequent analysis. You can use SonFix to repair such files.

The 64-bit smrx file system implements a data buffering scheme that allows us to delete pairs of duplicated times. This means that any even number of level events at the same time are deleted and an odd number are recorded as a single event. There is still the possibility of a problem with this scheme if a commit happens that writes a level event, then the next event is at the same time. As a future extension, it may be possible to keep pairs of level events at the same time by setting the second event one tick later; this would at least preserve the event if not the timing of it. This would probably need to be an option; most multiple events are not intentional.

Keep all events

You can also set the debounce period to -1 to ask the system to attempt to keep simultaneous events by giving all the simultaneous events consecutive clock ticks.

When to use this option

One obvious use of this option is if you get a message from sampling stating that there were multiple events at the same time and suggesting that you use this option. This warning only occurs if the repeated events are so quick that two or more get the same time stamp. It is more common that you have events that look correct, but the number of events counted between cursors is more than you expect. Zooming in on the data will reveal that what looked like single events were, in fact, multiple ones. A simple way to find these is to display the event channel as an instantaneous frequency.

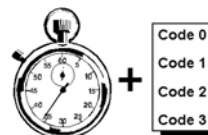
TTL compatible signals

TTL stands for Transistor-Transistor Logic, a method for passing logical information between devices using voltage levels. Levels above 3.0 Volts are in the High state, levels below 0.8 Volts are in the Low state. Levels in between 0.8 and 3.0 Volts are undefined.

Do not subject 1401 TTL inputs to voltages above 5.0 Volts or less than 0.0 Volts. CED hardware has special circuits on TTL compatible inputs to provide some protection, however determined abuse will damage them. The 1401 TTL compatible inputs are pulled up by a resistor to 5 Volts. They require a current of no more than 0.8 mA to pull them into the Low TTL state. Alternatively, you can connect them to ground to pull them low (this can be useful for the Trigger input). See the *Owners handbook* of your interface for full details of all input ports.

Marker data

Spike2 samples keyboard markers on channel 31 and digital markers on channel 32. A Marker is a 64 bit time plus 4 bytes of marker information (codes). The first of these 4 bytes is the ASCII code of the keyboard character pressed by the user (channel 31) or an 8 bit digital code read by the 1401 (channel 32). The remaining three bytes are set to zero. There is also an additional 4 bytes that are reserved for future use, either as 4 more marker codes, or as an addition 32-bit value.



Spike2 treats all marker types identically once the data has been captured; they differ only in their source. You can also treat data types that are derived from markers (such as WaveMark, TextMark and RealMark) as if they were markers.

Keyboard markers

The timing of Keyboard markers is not precise; it depends on the load on the computer. Use the event inputs for exact timing. Any keyboard character that is not trapped for a special purpose (for example `CTRL+L` opens the Evaluate windows) is recorded, but *only when the sampling document window is the current window*. In the special case where keyboard markers trigger data sampling, the precise time of the trigger is stored. You can link keyboard markers to the output sequencer. The keyboard marker channel is always enabled.

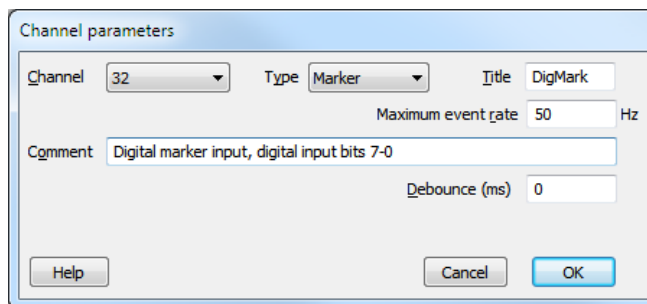
Digital markers

The digital markers are timed as accurately as the event data. They record 8 separate channels of on/off information, or one channel of 8 bit numbers, or any combination in between. Digital marker data is sampled when a low going TTL compatible pulse is detected as described below. The data is read from bits 7-0 of the 1401 digital input.



Digital marker dialog

The Maximum event rate field is used to allocate the system resources for data capture on this channel. Set a reasonable estimate of the maximum sustained data rate over several seconds. Do not set the peak rate or you will waste resources.



The Debounce (ms) field sets the minimum period between digital marker events with the same marker code that you expect and is used to reject spurious markers caused by bouncy mechanical switches. Markers with different codes can be spaced closer than this time as long as the time difference is not 0. You can also set a negative value (-1 recommended) to assign markers at the same time to consecutive clock ticks. Set this to 0 to disable this feature (which saves time on every Marker).

Digital marker connections

Marker data bit	7	6	5	4	3	2	1	0	Gnd
Digital input pin	5	18	6	19	7	20	8	21	13

The digital marker is read from data bits 7-0 of the digital input connector. The Micro1401 and Power1401 require a pulse on digital input pin 23 to flag a digital marker. In addition to the data lines, there is an optional handshake (h/s) signal.

Signal	Micro & Power
Event flag	pin 23
h/s	pin 24
h/s active	TTL low

To flag an event, apply a low going TTL pulse at least 1 μ s wide to the Event flag input. When the 1401 detects a falling edge at the Event flag input, it sets the h/s line active (within a few microseconds). The falling edge of the Event flag input latches the input data in the Micro1401 and the Power1401. The h/s returns to a non-active state after the 1401 reads the input.

Output sequencer link

There are links between the digital marker channel and the output sequencer. If the `REPORT` instruction is used in an output sequence, this simulates a digital marker input pulse and causes the digital input to be read and the time to be recorded. As this is an internal activity, the handshaking described here is not available.

You can also use the `MARK` output sequence instruction to record a digital marker without reading the digital inputs (the instruction sets the 8-bit marker code). This instruction is often used to record output sequencer actions as part of the data file.

You can mix externally and internally generated digital markers, but this is not recommended unless care is taken to differentiate between the two sources of markers during analysis. This could be done by connecting the external marker handshake line to one of the digital marker data bits so that all external markers were flagged.

Warning: The `DIBEQ`, `DIBNE`, `DIGIN` and `WAIT` sequencer instructions use the same inputs as the digital marker and can cause digital marker events to be missed. Spike2 will warn you if this is an issue with your hardware.

Marker codes

When Spike2 displays markers, or data derived from markers, such as WaveMark or TextMark data, it shows the code of the first of the four markers. Marker codes occur in several other guises, for example to set trigger codes, as arbitrary waveform output codes and in the spike shape module.

Marker codes have values from 0 to 255. This is the same range of numbers that the 8-bit ASCII character set uses, and it is sometimes convenient to treat the codes as ASCII character codes (for instance when dealing with keyboard markers). At other times it is more convenient to deal with the codes as numbers.

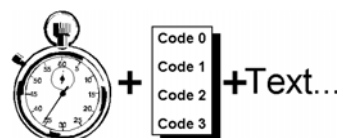
Whenever Spike2 displays a marker code that is the same as the ASCII code of a printing character, it shows the printing character, otherwise it displays the character as a two digit hexadecimal code. Hexadecimal (base 16) numbers use the standard digits 0 to 9, but also use a to f (for decimal 10 to 15). Thus 00 to 09 hexadecimal is equivalent to 0 to 9 decimal. 0a to 0f is equivalent to 10 to 15 decimal. 10 to 1f hexadecimal is 16 to 31 decimal, 20 to 2f is 32 to 47 decimal and so on. This behavior can be changed so that only hexadecimal codes are displayed on a channel by channel basis using the Draw Mode dialog or with the `MarkShow()` script command.

+	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
20	!	"	#	\$	%	&	'	()	*	+	,	-	.	/		The printing characters are 20 to 7e hexadecimal, 32 to 126 decimal, as in the table. Character 7f may or may not print, depending on the character set. To find the hexadecimal code of a printing character, add the number above it to the number to the left of it. For example, the code for A is 41. To convert a code to a character, look up the first digit in the left column and the second in the top row. For example, 3f codes to ?, the intersection of the row for 30 and the column for f.
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_	
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~		

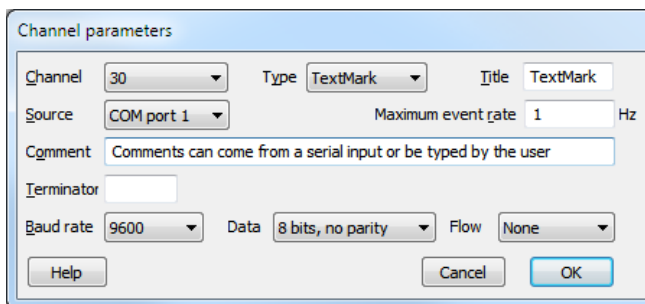
When typing marker codes (for example in the on-line Process dialog in Triggered mode, or when assigning codes in the spike shape module), type two hexadecimal digits for a code or type a single character to stand for itself.

TextMark data

This type is a combination of a marker and a text string. It is stored as a 32-bit time and 4 bytes of marker information followed by a text string that can be up to 80 characters long. This type allows you to insert timed comments into a data file. In the special case where the TextMark channel triggers data sampling, the precise trigger time is stored. From a script you can set longer or shorter strings with the `SampleTextMark()` command.



The channel number is forced to 30. If you select a channel number of 30, the data type is forced to TextMark. The Maximum event rate sets the expected maximum TextMark rate over several seconds and is used to allocate sampling buffers.



Serial line input

The Source field can be set to Manual or a choice of serial ports. If you select a serial port more fields can be set. The Terminator field sets the input character that marks the end of a text line. This field uses the same coding as for marker codes, so a single character stands for itself; two characters are interpreted as a hexadecimal code. Common codes are 0D for carriage return (the default) and 0A for line feed; 00 cannot be used. If no terminator is set, 0D is used.

You can also set the standard serial line parameters for Baud rate, data bits and parity and handshaking. These must match the data source for reliable operation. See your computer hardware manual for pin connections and Baud rate limits.

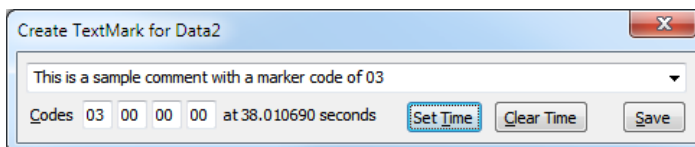
When reading the serial line, characters codes below 32 are ignored unless they match the terminator. The marker time is set when the first character arrives except for the special case where the TextMark channel triggers data sampling, when the precise trigger time is stored. The serial data can set the marker code by ending the text with a vertical bar followed by a decimal or hexadecimal number that encodes the marker codes c0, c1, c2 and c3 as: $c0 + 256*(c1+256*(c2+256*c3))$. Each code is in the range 0 to 255.


The vertical bar and the following text are excluded from the recorded data. If <term> stands for the terminating character, all the following are acceptable inputs:

```
Message from serial, all codes are 00<term>
Marker code 0=0e (decimal 14), codes 1-3=00|14<term>
Set code 0=01, 1=02, 2=03, 3=00 as hexadecimal|0x30201<term>
```

Manual input

TextMark data is added to your file during sampling from a serial line and with the Sample menu Create a TextMark... command. You can also activate the dialog with the Ctrl+T key combination as long as the sampling data file is the active window.



 TextMark data is drawn as small rectangles. The rectangles are yellow unless the first marker code is non-zero, in which case the same colour coding as for WaveMark data is used. Move the mouse pointer over a marker to see the attached text. Double click to view and edit the text and codes and display a list of the markers in the file.

If you enable this channel, Spike2 logs any programmable signal conditioner changes as TextMark items.

High TextMark sample rates

TextMark data was not (originally) designed with high sample rates in mind. However, some users have taken to using this channel type to store large quantities of data at relatively high sample rates. Some steps were taken at version 7.11c to make this work better, and more changes have been made in version 8. If you have large numbers of TextMark data items in a file we limit the number of displayed items in an attempt to maintain performance. If you are using an old .smr file with huge numbers of TextMark items, this dialog may be noticeably slow to open.

RealMark data

RealMark stores a 64-bit time, 4 bytes of marker information, then a user-defined number of single precision floating point numbers. You can create and manipulate this data type via a Talker, from the script language or by using active cursors to process measurements to a channel or through memory channels.

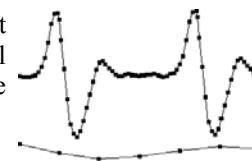
If you import waveform data into a RealMark memory channel, the first of the floating point numbers is set to the peak, trough or level value that was used to detect the event.

RealWave data

This data type was added at version 4.03. It is identical to waveform data except that the data is stored as 32-bit IEEE floating-point data, not as 16-bit integers. The channel has a scale and offset. These are used to convert between waveform data and RealWave data:

$$\text{RealWave data} = \text{integer data} * \text{scale} / 6553.6 + \text{offset}$$

You cannot sample RealWave data; you can create it as a memory or virtual channel and with scripts. Spike2 versions before 4.03 cannot open data files holding this data type.



Using RealWave data as integers

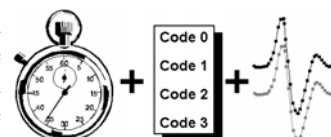
There are several places in Spike2 where you can specify RealWave data for implicit use as an integer. For example, if you specify a RealWave channel as a source for data written to a DAC for playing offline or for an arbitrary waveform. In these cases, the channel scale and offset are used to convert the data:

$$\text{integer data} = (\text{RealWave data} - \text{offset}) * 6553.6 / \text{scale}$$

If the result would exceed the range -32768 to 32767 it is limited to these values.

WaveMark data

This type combines waveform and marker data. It is stored as a 64-bit time and 4 marker bytes, followed by up to 126 waveform points on 1, 2 or 4 traces. The traces hold a spike shape. The first marker byte holds the spike classification code or 0 if it is unclassified. Script users can create WaveMark data for use off-line with up to 1000 data points.

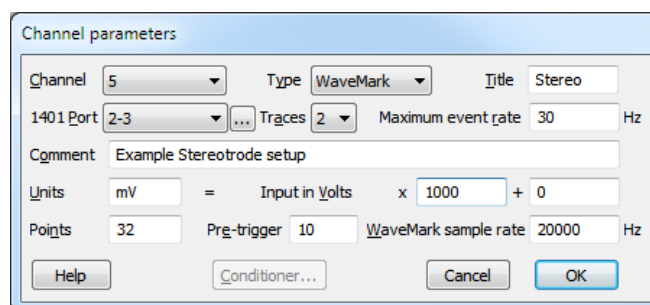


Use WaveMark data where a high waveform sampling rate is needed to characterise very short events, for example nerve spikes. When the incoming waveform crosses a trigger level, the signal is tracked to the next peak (or trough), and data around the peak is saved.

WaveMark dialog

The Maximum event rate is the expected maximum spike rate averaged over several seconds. The WaveMark sample rate field sets the waveform rate for all WaveMark channels.

The Units field is the same as for a waveform channel. You can use the Analysis menu Calibrate command to calibrate known values. There are fields to set the number of data and pre-trigger points per event; these can be adjusted during template formation.



Points The waveform points per trace to store for each WaveMark on this channel. You should set this to the smallest value you can (the larger the value, the more space is used on disk, and the slower it is to process).

Traces A Micro1401 mk II or -3 or a Power1401 can sample multiple traces for stereotrode and tetrode data. By default, traces use consecutive ports; the 1401 Port field sets the port for the first trace. For example, if the 1401 Port field is set to 2 and you have 4 traces, data will be sampled on 1401 ports (ADC inputs) 2, 3, 4 and 5.

However, with multiple traces you can click the ... button to open a dialog where you can set non-sequential ports. The first port set for each WaveMark channel must be different, the following ports are a free choice.

Pre-trigger The number of data points to keep before the first peak or trough to exceed the trigger level for this channel.

WaveMark This field sets the ideal sample rate for all WaveMark channels. Spike2 will adjust the sampling

sample rate parameters to get as close to this rate as it can. See the Resolution Tab description for more information about rates.

The total number of WaveMark traces you can sample depends on the 1401 type: 32 for Power1401 and 16 for Micro1401 mk II. For example, a Power1401 could sample 4 WaveMark channels with 4 traces plus 4 with 2 traces plus 8 with one trace.

The Conditioner... button is enabled if a programmable signal conditioner is present.

Spike sorting

Spike2 can match waveforms to a set of templates. This is normally used to extract single spike units from multi-unit recordings, but other uses are possible (for example extracting R waves from ECG data). If you record WaveMark data, a template setup window appears when you open a file for sampling.

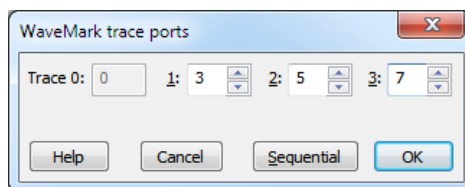
Set WaveMark codes

When you display WaveMark data as a waveform in a time view you can change the marker codes of items that you select with the mouse. Hold down **Alt+Ctrl** and click and drag a line over the events you want to identify. On mouse up, a dialog opens in which you can set codes for the intersected events.

Non-sequential ports for traces

From Spike2 version 8, the ports used for tetrode and stereotrode data need not be sequential. If you click the ... button in the WaveMark setup dialog when 2 or 4 traces are active you can use this dialog to set the ports.

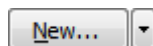
The port for trace 0 is fixed (it is set by the WaveMark dialog). The remaining ports are a free choice; there is currently no check that these are different from each other. The **Sequential** button sets the ports into sequential order.

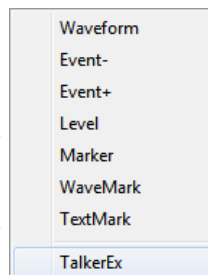


Talkers

You can sample additional data channels from plug-in devices, called Talkers. Each Talker can provide one or more channels of data; these channels can be of any data type that Spike2 supports. A talker can also provide additional keyboard marker channel input. Note that sampling data from Talkers is not a replacement for sampling with a 1401 but an additional capability; Spike2 always requires a 1401 in order to sample data.

For example, suppose you have a blood pressure monitor that automatically inflates a cuff every 10 minutes and measures systolic, diastolic and mean blood pressure. If this device has a computer interface, the output could be added automatically to a Spike2 data file by writing a Talker interface. It could either provide three channels of RealMark for the three outputs, or it could provide a single channel of RealMark data with three attached data items.

 Channels from Talkers are added to Spike2 from the Channels tab of the Sampling Configuration dialog by clicking on the small button to the right of the **New...** button. A drop-down list of possible channel types appears, with any Talkers known to the system at the bottom. If you select one of the normal channel types, this is equivalent to clicking the **New...** button, then choosing a channel type. However, if you select a Talker name, a new Talker dialog opens.

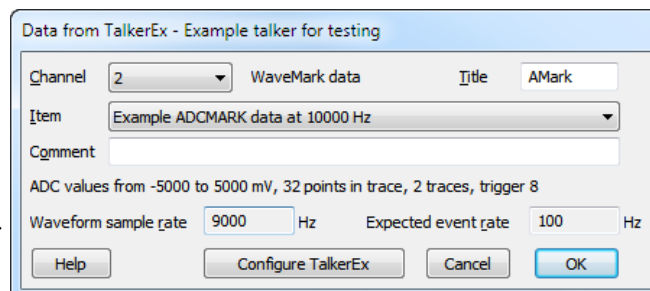


If any item in the drop-down list is disabled, this means that the available quota of channels for that channel type or Talker has been used, or it can mean that the Talker only generates keyboard marker items, so cannot be used to add a new channel to the sampling configuration. If a Talker is not currently connected to Spike2, the talker name is followed by an exclamation mark (!) to warn you that you cannot sample with it until it is connected. A talker that only generates keyboard markers is not listed if it is not connected.

There are additional Talker controls available from the Sample menu Talker list command.

Talker dialog

For the purposes of this example we will assume that you have selected a Talker called TalkerEx that we provide with Spike2 as an example. You will find this Talker as TalkerEx.exe in the Talkers folder in the folder where Spike2 is installed. Just double-click it to set it running, then select TalkerEx from the drop down list next to the New... button in the Channels tab of the Sampling Configuration dialog.



When you select a Talker in the Sampling Configuration dialog by clicking on an enabled talker name, a new dialog opens. The editable fields are:

- Item** Every Talker that can generate this dialog provides one or more data channels to write to the data file. This field selects the channel to add to the sampling configuration. When you add an Item, it is removed from the list of available items for this Talker. Once all available items have been added for a Talker, the Talker name is disabled so you cannot add further channels.
- Channel** This field sets the channel in the Spike2 data file to use to store the Talker data. You can change the channel number, but not to a channel that is already in use.
- Title** Up to 8 characters to identify the channel.
- Comment** Text to give more information about this channel.

The remaining fields are not editable and give information about the channel. The values displayed here are supplied by the Talker.

Configure ...

The Configure TalkerEx button is enabled if the Talker is currently connected and if it supports configuration. It opens a dialog that allows simple parameters of a Talker to be changed. It is likely that complex talkers will provide their own setup dialog system that runs as a separate entity from Spike2. However, if a talker is hosted on a remote computer, it may be easier to allow simple configuration from the computer running Spike2. This can allow multiple Talkers attached to different computers to be controlled from a single computer.

Talker documentation

As Talkers are independent of Spike2; apart from the three example talkers described here, you will not find documentation for specific Talkers in the Spike2 documentation. A Talker that is written by CED to support specific hardware will have its own documentation, and we would hope that any Talkers provided by third parties to support their hardware will be suitably documented. If you have a problem with a non-CED originated Talker, it is unlikely that we will be able to provide you with support; you will need to take this up with the authors.

Spike2 keeps a list of all Talkers that it has been exposed to in a repository called SP2TALKS.DAT in the folder where Spike2 runs from. This allows Spike2 to configure sampling for talkers that are not currently connected.

Communications between the Talker and Spike2 are done using *named pipes*. This allows the Talker to run on the same computer as Spike2, or on a separate computer linked by a network. There is documentation and example code available to allow anyone who wishes to write their own Talker to connect to Spike2.

You can download the Talker information and example source code from our web site.

Example Talkers

We include three example talkers with Spike2 in the `Talkers` folder in the same location where Spike is installed. You can run them on the same computer as Spike2 by double-clicking them. If you have a local area network connecting your computers, you can run them on a different computer by using a command line parameter:

```
C:\PathToTalker\TalkerName -sComputerName -n1
```

Where `C:\PathToTalker\` represents the location of the talker and `TalkerName` is the name of the talker. The `-s` option sets the name of the server computer that is running Spike2. If you omit the `-s` option, the server is assumed to be running on the same computer as the Talker. Talkers are identified by name and if you have multiple talkers, they must have different names. The `-n` option allows you to run multiple identical talkers; the number immediately after the `n` is appended to the Talker name, so `TalkerEx -n1` generates a talker called `TalkerEx1`. If the computer running Spike2 is called `Samuel` and your talker is in the folder `C:\Talkers`, you can connect with:

```
C:\Talkers\TalkerEx -sSamuel
```

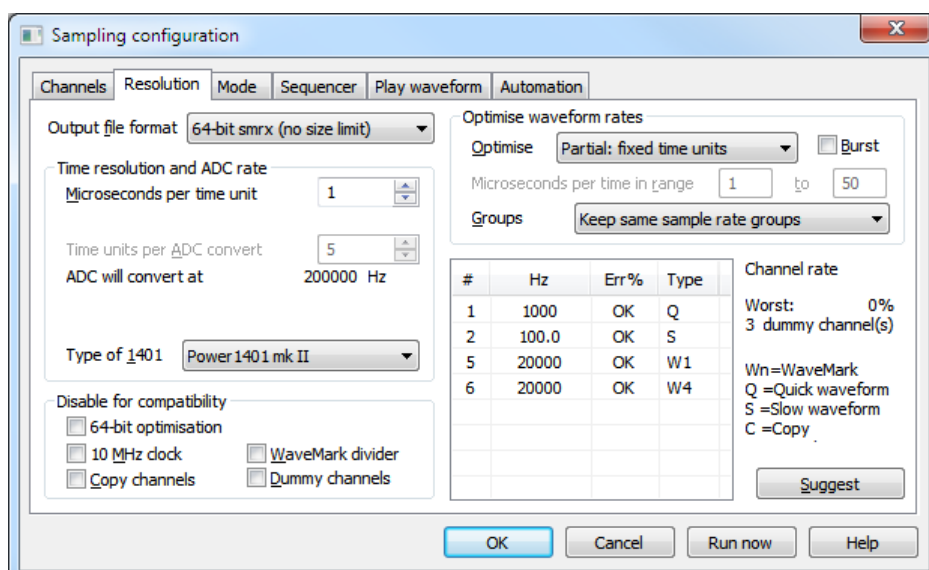
The three example talkers (source code is available on our web site) are:

- `TalkerEx` Provides dummy channels of every type and generates dummy data. It also logs keypresses and sends them to the keyboard marker channel. This is purely an example; it has no practical use.
- `MKeys` Provides an additional marker channel that logs keypresses. This acts as a remote keyboard, so only makes sense if run on a different computer. You could use this to log responses from multiple subjects.
- `XKeys` Provides additional input to the keyboard marker channel. This acts like a remote keyboard, so only makes sense if run on a different computer. You could use this to log responses from a single subject.

Resolution

The **Resolution** page of the sampling configuration dialog sets the time resolution of the document and the ADC sampling rate. To get the best possible results, you should read all this section. However, to get started quickly, follow these “cookbook” instructions:

1. Set the **Output file format** field (set 64-bit `smrx` unless your files need to be read by old software).
2. Set the **Type of 1401** field to the 1401 type you will use.
3. Set **Optimise** to **Partial: fixed time units**.
4. Set **Groups** to **Keep same sample rate groups**.
5. If you selected a 32-bit file type in step 1, set **Microseconds per time unit** so that the **Longest run time** field is at least as long as the time you want to sample for. Otherwise, set **Microseconds per time unit** to the smallest value that gives you the waveform sample rates you need.
6. Clear all the **Disable for compatibility** section check boxes (the state of disabled boxes does not matter).
7. If the **Burst** check box is visible, set it unchecked unless you know that you need burst mode.



This dialog controls how Spike2 optimises the sampling rates. Spike2 minimises the sum of the proportional errors between the desired and the actual rates for the waveform and WaveMark channels. By proportional we mean that an error of 200 Hz in a sampling rate of 10 kHz is the same as an error of 2 Hz in a rate of 100 Hz.

Output file format

Spike2 version 8 is optimised to create and work with 64-bit `.smrx` data files. Unless you have a requirement to use 32-bit `.smr` files (for example because you use third-party programs that do not read `.smrx` files to analyse your data or your collaborators are using a version of Spike2 that cannot read `smrx` files) you should set this field to 64-bit. See *Data file types*, below, for full details. The script language equivalent of this field is `SampleBigFile()`.

Type	Name	Comments
64-bit	*.smrx	64-bit data files that are limited in size only by the capabilities of the operating system and have no time limitations. This allows both high timing resolution and long run times. Spike2 version 8 is optimised for this format. Use this unless you need backwards compatibility.
32-bit big	*.smr	32-bit data files limited in size to 1TB and in time to 2 billion clock ticks. This is compatible with Spike2 version 7 and can be read by version 6.11 onwards.
32-bit	*.smr	32-bit data files limited in size to 2 GB and in time to 2 billion clock ticks. This is the most compatible format for old versions of Spike2.

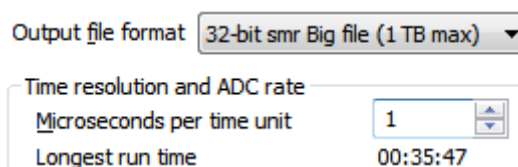
Microseconds per time unit

This field sets the time units for a new data file in the range 1 to 10000 microseconds. All data of any type stored in the file occurs at a multiple of this time unit. If you have selected a 64-bit file you will want to set this to the lowest value possible (giving the best time resolution) compatible with the desired sampling rates. If you have selected a 32-bit file, you will have to balance the desire for timing accuracy against how long you need to sample for, see below. To edit this field, set **Optimise** to **None** or **Partial**. If you select a Power1401 or Micro1401 mk II or -3 in the **Type of 1401** field, you can set this field to a resolution of 0.1 instead of 1 microsecond.

Setting the resolution to non-integral microseconds prevents 32-bit files being read by versions of Spike2 prior to 4.02. 64-bit files cannot be read by versions prior to 7.11c.

Longest run time

This field is visible if you selected a 32-bit file in the **Output file format** field. With a 32-bit file, as in the example to the right, the maximum sample time is 2,147,483,647 ticks of the **Microseconds per time unit** field and the **Longest run time** field displays this duration in years, days, hours, minutes and seconds for the current



time units. If this period is shorter or longer than you require for your work you can adjust the **Microseconds per time unit** field to give you the duration you need.

There is a length limit for 64-bit files but it is so large (256 thousand years at 1 microsecond resolution) that it is of no practical concern.

Time units per ADC convert

This sets the ADC (Analogue to Digital Converter) clock interval in the units set by **Microseconds per time unit**. Each time the ADC is clocked either one sample is taken (non-burst mode) or a group of samples is taken (burst mode). To edit this field set **Optimise** to **None**. The **ADC will convert at field** is the equivalent rate in Hz for normal mode; it changes to **ADC burst rate** in burst mode.

Type of 1401

Members of the 1401 family have different capabilities. Although you cannot use the *1401plus* or the original *micro1401* to sample, you can still select them in the list in case you need to match previous settings. The choice you make here sets the absolute maximum settings for sampling.

There is no guarantee that the maximum settings are achievable. The maximum rate depends on the entire sampling configuration, the input data load, the speed of the 1401, the speed of the interface connection and on the capabilities of the host computer. For example, if your 1401 is plugged into a USB 1 port (or has a USB 1 interface), the maximum continuous sample rate to the host is 400 kHz (or less), which will limit your aggregate ADC rate to a maximum of 200 kHz. You can set:

General compatibility

Any 1401 except a *1401plus* with the old analogue card (an upgraded standard 1401). This is a *lowest common denominator* setting with a maximum waveform sampling rate of 166 kHz and does not assume that your monitor firmware is the most recent. We retain this setting only for reasons of backwards compatibility.

1401plus, old ADC (not supported for sampling)

If your *1401plus* was upgraded from a standard 1401 without an upgrade of the analogue card you must select this option. This option is preserved so that you can match old sampling rates.

micro1401, 1401plus (not supported for sampling)

This setting is for a *micro1401* or a *1401plus* with an up-to-date monitor. You can find if there is more recent firmware available in the Help menu **About Spike2** command. This option is preserved so that you can match old sampling rates.

Power1401, Power1401 625 kHz

These settings are for the *Power1401*. You can set **Microseconds per time unit** in units of 0.1, and **Time units per ADC convert** to 1 and sets the maximum ADC convert rate to 400 kHz (625 kHz with the later *Power1401 625 kHz*). Selecting any *Power1401* or the *Micro1401 mk II* enables additional optimisations when calculating waveform rates (*10 MHz clock*, *WaveMark divider*, *Copy channels* and *Dummy channels*).

Micro1401 mk II, Micro1401-3

Select this option to take advantage of all the capabilities of the *Micro1401 mk II* and *-3*. This allows **Microseconds per time unit** to be set in units of 0.1, **Time units per ADC convert** to be set to 1 and sets the maximum ADC convert rate to 500 kHz.

Power1401 mk II, Power1401-3

This setting allows **Microseconds per time unit** to be set in units of 0.1, **Time units per ADC convert** to be set to 1 and sets the maximum ADC convert rate to 1 MHz.

Disable for compatibility

If you replace a 1401 with a more recent model or upgrade Spike2 from before version 6.05 you may get different waveform and WaveMark sampling rates. The new rates will be closer to the requested rates, but if you are half way through a study, compatibility with earlier data may be more important. This section of the dialog disables new features so you can match the original rates. Fields that do not apply to the device selected in the **Type of 1401** field are disabled. The settings of disabled fields are ignored. The fields are disabled for the *1401plus* and the original *micro1401* as these devices do not support any of the following features.

64-bit optimisation

Check this box to disable new optimisations in version 8 that take advantage of 64-bit times. With a 64-bit file, when searching for acceptable Microseconds per time unit values, we take the lowest value that minimises sampling rate errors. With a 32-bit file or with this option selected, we take the highest value.

10 MHz clock

Check this box to force the Microseconds per time unit to be an integral number. Before version 6.05 this control was available as part of the Groups field as *1 MHz, same sample rate groups*.

WaveMark divider

Previously, if you had WaveMark (spike shape) channels, the sampling rate of the WaveMark data set the maximum sample rate for all other waveform channels. Now, we allow waveform channels to be sampled faster than WaveMark channels by taking 1 in every n WaveMark points (dividing the WaveMark rate by n).

Copy channels

If you sample the same channel as both a waveform and as a WaveMark, we only sample the data once for the WaveMark channel and take a copy for the waveform channel. Check the box to sample each channel separately.

Dummy channels

With older versions of Spike2, it was sometimes possible to get a better match to the desired sampling rates by adding more channels and then ignoring the data in them. For example, if you requested 7 waveforms at 10 kHz each you got 9.99 kHz. Adding another waveform channel produced 10 kHz again. Dummy channels do this for you automatically, and do not waste any time moving data back to the host. Dummy channels are not used in burst mode.

Optimise

The Optimise field sets the parameters Spike2 changes to minimise sample-rate error:

None: use manual settings

You have control over the values in the Microseconds per time unit and Time units per ADC convert fields. In this mode click the Suggest button to change the fields to the values that minimise sample rate errors.

Partial: fixed time units

You set the Microseconds per time unit field and Spike2 adjusts the Time units per ADC convert field to minimise the sampling rate errors. If there is more than one solution, Spike2 chooses the one with the slowest ADC convert rate.

Full: time units can change

Spike2 sets the Microseconds per time unit and Time units per ADC convert fields. If all channels have a slow rate, this can take a long time, especially in burst mode. The Microseconds per time range fields set the acceptable range of time units. If there are multiple solutions, Spike2 chooses the one with the biggest Microseconds per time unit.

Groups

This field places additional restrictions on how Spike2 maps the requested sample rates for all the waveform and WaveMark channels into achievable sampling patterns.

Version 3 compatible

This gives the same waveform rates for a given Microseconds per time unit and Time units per ADC convert as version 3. Only use this if you upgrade from version 3 and it is vital that sampling rates match old data files.

Keep same sample rate groups

If you select this option, Spike2 will make sure that all waveform channels with the same ideal sampling rate have the same actual rate. You will normally use this option.

Ignore same sample rate groups

This option gives the smallest sampling rate errors. The price you pay is that channels with the same ideal sampling rates may get actual sampling rates that are different. Some data analyses, such as waveform correlations, multiple channel averages and power spectra demand that channels have identical sampling rates.

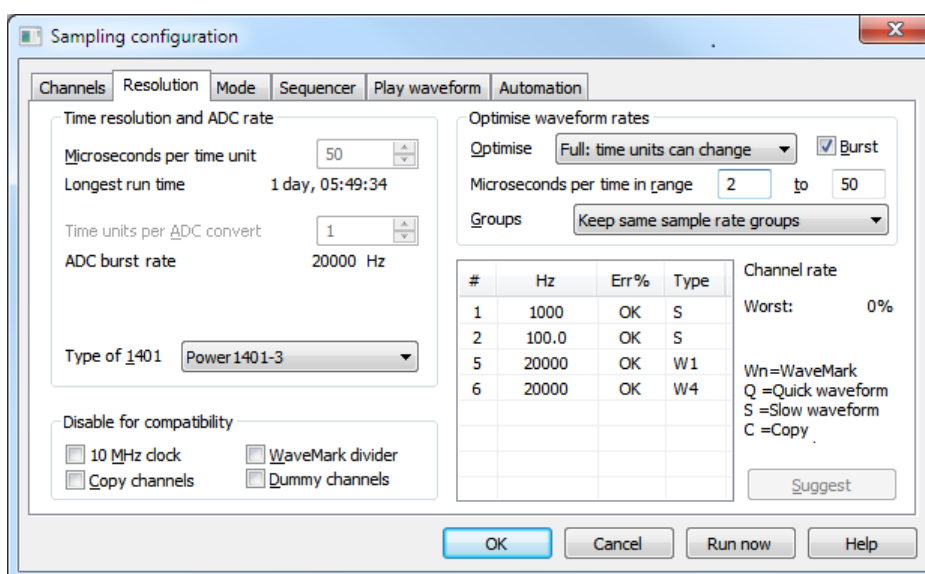
1 MHz, same sample rate groups

This setting was present before version 6.05 but no longer exists. You can achieve this now by disabling the 10 MHz clock in the **Compatibility** section and selecting *Keep same sample rate groups*. If you read an old configuration that used this setting it will be translated to work in the same way.

Burst mode

Burst mode sampling was added at version 6.06 and can be useful when you sample more than 1 waveform or WaveMark channel. If you have n waveform and WaveMark channels, the advantages of burst mode are:

- You may be able to run the Spike2 clock (set by Microseconds per time unit) n times slower. This allows longer sample durations if you sample to a 32-bit file; this is not a consideration for 64-bit files.
- You may be able to achieve sample rates per channel that are n times higher for the same Spike2 clock rate.



The disadvantage of burst mode is that waveforms are sampled at times that may fall between Spike2 clock ticks. Channels are sampled at precisely the correct interval, but each channel is shifted sideways by up to half a Spike2 clock tick from when it was sampled. In many applications this will not matter, and the benefits of a better sampling rate may outweigh this. In version 8, where you can have a very small clock tick, this may not matter at all. Burst mode is most effective when you have a large number of waveform and WaveMark channels.

The **Burst** check box is visible if you select any type of Power1401 or a Micro1401 mk II or -3 (which is usually the case in Spike2 version 8). If you check it, instead of sampling the ADC at equal intervals synchronised to the 1401 clock, it samples a burst of channels at a time, each burst being synchronised to the clock. When you check the Burst mode box, the ADC will convert at field changes to ADC burst rate.

If you set the **Optimise** field to *None*, for full manual control, you will normally set the **Time units per ADC convert** field to 1 as higher values tend to reduce the benefits of burst mode.

Data file types

Spike2 version 8 is optimised to use and sample data to the 64-bit SON64 filing system. It can also use and sample data into the original 32-bit SON filing system. We would recommend that you use the new format unless you have pressing reasons for using the old. Not all sampling features are supported by the old filing system and at some point we will remove support for sampling to it.

The 32-bit file system (*.smr)

The original Spike2 data format was first released as version 1 around 1988 and the original design has been extended several times since; we are currently at version 9. Each revision has added new data types and/or new filing system features. Change were incremental; older versions of Spike2 would still read newer files as long as they did not use new data types or features that the older versions did not know about.

The original design was based on the idea that all items in the file were located in time at an integer multiple of a basic clock period and that there are two fundamental types of data: waveforms (equally spaced in time data samples) and events (items at defined times with other data attached). There is some crossover between these types; waveforms can have gaps and events can have waveforms attached to them. You can download a manual explaining this library plus interface software from our web site.

32-bit original format

There are two limits to the size of a 32-bit Spike2 data file: the number of clock ticks in the file (limited to 2147483647) and the physical size of the file. The physical size before version 7 was limited to 2147483647 bytes, or 2 GB. These limits are related to the maximum size of a signed 32-bit number. For many users these limits are not a problem. However, it is possible to run out of disk space before you run out of clock ticks. For example, if you set the Waveform sample rate to be one clock tick and sample one channel, each sample point uses 2 bytes of data. If you set Burst mode sampling in the Resolution tab with n channels, each taking one sample per clock tick, then each clock tick is using $2n$ data bytes. This means that your maximum possible run time is limited to $2147483647/2n$ clock ticks.

Before Spike2 version 7, if you were sampling 32 channels of data at 31.25 kHz each, in non-burst mode you would need to set the clock tick to 1 microsecond, giving a clock tick limit of some 35 minutes of sampling. However, the file size limit would be hit in around 17 minutes. If you changed to burst mode sampling, the clock could be run at 32 microseconds per tick, which extends the clock limit to 19 hours, but the file size limit is still 17 minutes.

32-bit big files (Spike2 version 7 onwards)

If you set the file type to 32-bit big, the maximum file size is increased by a factor of 512, and with the example given above (32 channels at 31.25 kHz each in burst mode), you could sample for the full 19 hours allowed by the clock tick period. However, your timing resolution is now 32 microseconds. This would generate a huge file (128 GB) which would be slow to navigate.

32-bit limitations

The 32-bit system served us well, but has limitations due to the 32-bit nature of the format:

1. Times are stored as 32-bit integers, limiting us to a maximum of 2 billion clock ticks per file. At a resolution of $1\ \mu\text{s}$ (microsecond) the maximum duration is 35 minutes and 47 seconds.
2. The file size was originally limited to 2GB. This was later extended to 1TB, but at the cost of making data recovery harder.
3. It can take a long time to locate data in the middle of a file. Finding data was speeded up by including lookup tables, but the methods used were limited by the requirement for backwards compatibility.
4. Waveform data with gaps is not stored efficiently as each gap starts a new disk block, so short waveform fragments use a lot of disk space.

The 64-bit filing system (*.smrx)

The new 64-bit filing system was designed to be logically compatible with the 32-bit system; by this we mean that it can hold the same types of data as the original without information loss, though some of these types are extended. It is also likely that we will add further data types, as required, in the future. Features include:

1. Times are stored as 64-bit integers. At a time resolution of $1\ \text{ns}$ (nanosecond, 10^{-9} seconds), the maximum duration is around 256 years.
2. The file size is limited only by the capabilities of the operating system and by the size of a file that you can manage conveniently for archival. As I write this disk drives have a maximum size of a few TB. The file format is designed to make recovery of damaged files relatively straightforward.
3. The files have a built-in data lookup system designed to minimise the number of disk reads required to locate data on any channel.
4. The overhead for severely fragmented waveform data has been reduced to a few bytes per fragment.

We have removed many of the limits on things like the length of channels comments and units and the number of channels in a file. However, Spike2 does not (yet) take advantage of these features.

How older versions of Spike2 cope with 64-bit (.smrx) files

Spike2 version 7.11c onwards can read (but not modify) the new 64-bit file system if the `son64.dll` file is in the Spike2 version 7 folder. Of course, if the file is longer (in clock ticks) than the old file system can read, only the start of the file will be visible. No other older versions of Spike2 can read them.

How older versions of Spike2 cope with 32-bit (.smr) files

Spike2 5.15 onwards can read files with up to 400 channels. Versions from 5.00 to 5.14 read files with up to 256 channels. Version 4.03 onwards reads files with up to 100 data channels. Versions before 4.03 can read files with up to 32 channels. You can use the File menu Export command to write channels from a data file to a new file with a suitable maximum channel limit so that it can be read by older versions of Spike2. If you check the Big file box you will not be able to open any generated file with versions of Spike2 before 6.11. Spike2 version 6 revisions that can open the file will treat it as read only.

Technical details

A master clock in the 1401 controls all sampling. The Microseconds per time unit field sets the tick period of this clock. All times in a Spike2 data file are multiples of this time unit.

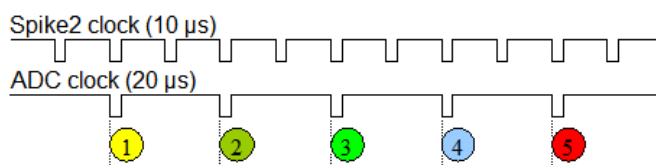
The Analogue to Digital Converter (ADC) samples one input at a time and is shared between all the waveform and WaveMark (spike shape) input channels. The table below lists the maximum sampling rate in multi-channel mode for the ADCs in each 1401 when used in Spike2 and also if the 1401 supports Burst mode sampling. Items marked * are not supported in Spike2 version 8 but are preserved for reference. Using a slow interface (for example USB 1.0) will reduce the maximum rates.

1401 type	Maximum rate	Can burst
1401plus*	166 kHz	No
micro1401*	166 kHz	No
Power1401	400 kHz	Yes
Micro1401 mk II/-3	500 kHz	Yes
Power1401 625	667 kHz	Yes
Power1401 mk II	1 MHz	Yes
Power1401-3	1 MHz	Yes

To illustrate the difference between non-burst mode and burst mode, we will consider what happens when we set Microseconds per time unit to 10 and Time units per ADC convert to 2 with five waveform channels in both non-burst mode and in burst mode. In the diagrams, the numbered circles represent the ADC sampling each channel and the horizontal position of the circle represents the time at which the sample occurs.

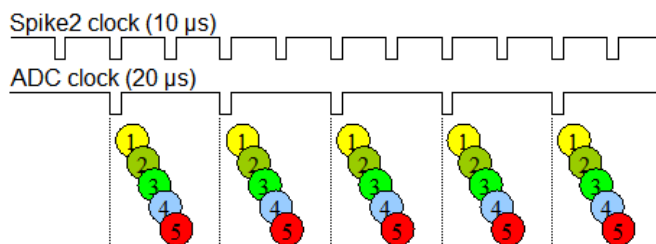
Non-burst mode

In non-burst mode, the Time units per ADC convert field sets how often the ADC samples in units of clock ticks. For example, with 10 microseconds per tick (but see *Improvements in version 8*, below) and the Time units per ADC convert field set to 2 (once every 20 microseconds), the ADC sample rate is 50 kHz. In a simple case with five waveform channels, the fastest each channel could be sampled is 10 kHz. Each sample is synchronised to the Spike2 clock and the pattern repeats every five samples.



Burst mode

In burst mode, the Time units per ADC convert field sets how often a burst of ADC samples is taken. With the same settings of 10 microseconds per tick and the Time units per ADC convert field set to 2, the burst rate is 50 kHz. In a simple case with five waveform channels, five channels would be sampled in a burst, each channel sampled at 50 kHz. The interval between samples in a burst depends on the 1401 hardware and is typically the interval implied by the maximum sampling rate for the 1401 hardware.



As you can see, the ADC samples no longer fall exactly at Spike2 clock times. The times between samples on a channel are exact, but the entire channel may be displayed time shifted by up to half a Spike2 clock. In this case, channels 1 and 2 would probably be timed for the tick just before them, and channel 3, 4 and 5 for the next tick.

Cycle of channels

To generate the sample rates, the ADC samples a cycle of channels. In burst mode all the channels in a cycle are sampled in a burst, in non-burst mode they are sampled one at a time. WaveMark channels are sampled once every time around the cycle. Some waveform channels are set as *Quick* and are also sampled every time round the cycle. The other waveform channels are set as *Slow*, and these share one position in the cycle. Quick and Slow channels save every *n*th data point (*n* in the divisor in the range 1 to 2147483647). The Version 3 compatibility setting in Groups sets all waveforms as Slow channels and the maximum divisor to 65535.

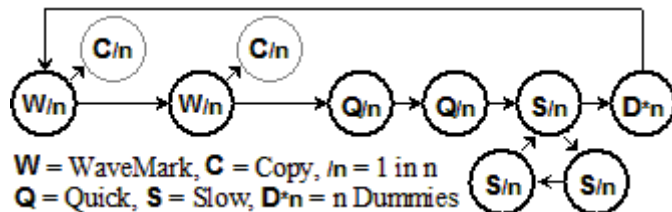
With a Micro1401 mk II or -3 or any Power1401, there are further features (unless Version 3 compatibility has been set):

WaveMark divider This enables down-sampling (taking 1 point in *n*) of WaveMark data. This allows you to sample Waveform channels faster than WaveMark channels.

Copy channel If you sample a waveform channel on the same 1401 port as a WaveMark channel, we can use the same data twice, once for the WaveMark channel and once for the waveform. These *Copy* channels behave exactly like a Quick channel, but are more efficient.

Dummy channel Adding additional channels to the sampling loop sometimes gives a more accurate approximation to the desired sample rates. Adding a Quick channel would waste time transferring unused data to the host; a *Dummy* channel just throws the data away. Spike2 adds dummy channels automatically if they improve the channel sampling rates. Dummy channels are not added in burst mode.

The diagram shows a possible cycle for two WaveMark channels and seven waveform channels of which 2 are the same as the WaveMark channels. Each time around the cycle, the 1401 samples all WaveMark, Quick and Dummy channels and one Slow channel. When waveform and WaveMark channels are sampled together, the fastest waveform rate is the same as the WaveMark sample rate.



Spike2 searches all ADC rates allowed by the Optimise setting and the Type of 1401 field for the best combination of Quick, Slow, Copy and Dummy channels and the best value of *n* for each channel to get as

close as possible to the ideal sample rates. With slow waveform rates there can be millions of combinations to search. If the Channels Tab feels very sluggish, set **Optimise** to **None**, set the channels, then restore the **Optimise** value. Impossible combinations display a warning in the lower left corner of the dialog.

If you check boxes in the *Disable for compatibility* section, this stops Spike2 taking advantage of optimisations that are not available for all 1401s. You might want to do this if you upgraded your 1401 and discovered that the sampling rates with your new 1401 were not the same as with the old one. Of course, the new rates would be closer to what you had asked for, but it might be important that they matched the old rates exactly.

The table to the left of the **Suggest** button lists the channels in order of descending sample rate error, showing the actual sample rate, the error as a percentage of the desired rate or OK if there is no error, and the channel type (WaveMark, Copy, Quick or Slow) and the number of Dummy channels.

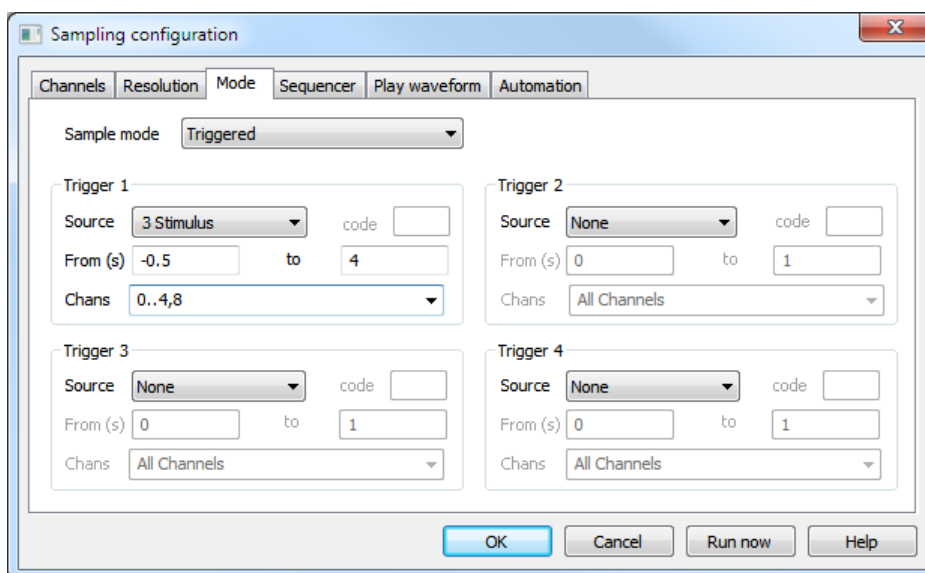
Improvements in version 8

Version 8 of Spike2 allows you to use 64-bit data files in place of the 32-bit files used in all previous versions. For 32-bit files, the maximum sample time is 2,147,483,647 clock ticks (2×10^9). At 2 microseconds per tick this is 71½ minutes, at 10 this is almost 6 hours, and at 1000 this is nearly 25 days. For 64-bit files, the maximum sample time is 8,070,450,532,247,928,832 clock ticks (8×10^{18}). This is so large that we need not worry about running into the limit. Several of the limitations in sampling in previous versions of Spike2 were caused by the requirement to maximise the **Microseconds per time unit** field to give long run times; selecting a 64-bit file format removes these limitations in version 8.

In the examples above, the Spike2 clock is shown as running at 10 us per tick, which is a commonly-used tick rate as it allows a run time of around 6 hours per data file, but at the cost of restricting the ADC sample rate, or forcing the use of burst mode. If you choose a 64-bit data file, you can usually run the Spike2 clock at 1 microsecond per tick, with a much wider choice of sample rates and no limitations on run time.

Sampling mode

The **Mode** page of the sampling configuration dialog determines when data is captured by Spike2 and saved to the disk system. Whichever sampling mode you select, the **Sample control bar** lets you disable data saving to the data document while sampling. Script users can enable and disable data saving channel by channel. In all modes, time passes at a constant rate, even when nothing is written to disk. When reviewed, areas of a file with no saved data are empty. There are three sampling modes: **Continuous**, **Timed** and **Triggered**. The script language equivalent of this dialog is the `SampleMode ()` command.



Continuous mode

The simplest sampling mode is **Continuous** mode, which records data continuously.

Timed mode

In Timed data capture mode, data is saved at intervals. You set the period for which data is saved and how often to save the data. The times at which the blocks were requested are saved in the keyboard marker channel. Marker code 00 is placed at the start of each timed block and marker code 01 is placed at the end of each block. If you use a 32-bit output file you should read the *64-bit and 32-bit file differences* section, below.

Triggered mode

In Triggered capture there are four triggers. Each has an associated channel list. A trigger is an event or marker that causes data to be marked for writing to disk in a time range relative to the trigger. Triggers are additive, that is if a new trigger occurs while data is being written due to a previous trigger, the trigger period is extended. In triggered mode, any channel that is not associated with a trigger is recorded continuously. Each trigger has the following fields:

- Source** This should be set to **None** to disable the trigger or you can select a channel from the drop down list. You can trigger on any event, Marker or WaveMark channel in the sampling configuration.
- Code** If the trigger source is a Marker, WaveMark, RealMark or TextMark channel, you can choose to trigger only if the first marker code matches this field. You can set two hexadecimal digits to set the marker code or one printing character or leave the field blank for a trigger on all codes.
- From** This value is the offset from the trigger to the start of the area to record, in seconds. Negative values define pre-trigger start times, positive values start recording after the trigger. If you set too long a pre-trigger time, the data may have been discarded before the trigger is seen. Spike2 normally attempts to use an 8 MB data buffer, so this is not often a problem.
- To** This value is the offset from the trigger to the end of the area to record, in seconds. It must be greater than the time in the From field.
- Channels** You can either select **All Channels** from the drop down list, or type in a list of channels, for example 1..4,6,8 for channels 1, 2, 3, 4, 6 and 8. These are the channels that will be written to disk each time the trigger event is detected.

Triggered sampling is usually used with fast waveform or WaveMark channels to save disk space in situations where only small sections of the data are interesting. However, if you use a 32-bit output file you should read the *64-bit and 32-bit file differences* section, below.

Special keyboard trigger features

If the user changes the state of the Write check box in the **Sample** control toolbar, a marker code is written to the Keyboard channel to indicate that writing to disk is enabled (code 00) or disabled (code 01). If the keyboard channel is used as a trigger, writing these codes does not trigger sampling.

Display during data capture

In all data capture modes, the on-line display shows newly sampled data, even when you are not saving to disk. Recent data is saved in a memory buffer in these modes so the current data is always available. However, if you scroll far enough back into an unsaved time region, the display may become blank.

64-bit and 32-bit file differences

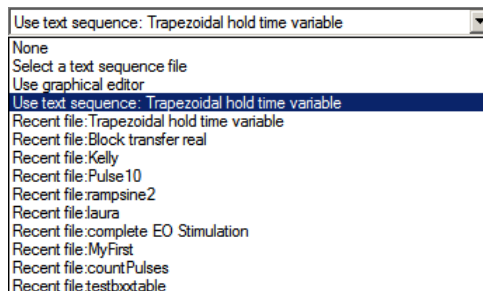
If you use triggered or timed mode with the 64-bit filing system, triggered or timed channels will save only the data implied by the timing or the trigger, as you would expect. Note that Level event channels are always saved in their entirety, regardless of the saving state, to ensure we track the levels correctly. This may be revised in a future revision by adding code to make sure that an even number of edges are dropped.

However, the 32-bit system can only save complete blocks of waveform data and each block is around 16000 data samples. If you select a 32-bit file output file, you will always save at least the data implied by the trigger or timing, but with waveform channels you will usually get more data than asked for. Further, if your triggers or timing periods are closer together than 16000 data points, you will save continuous data.

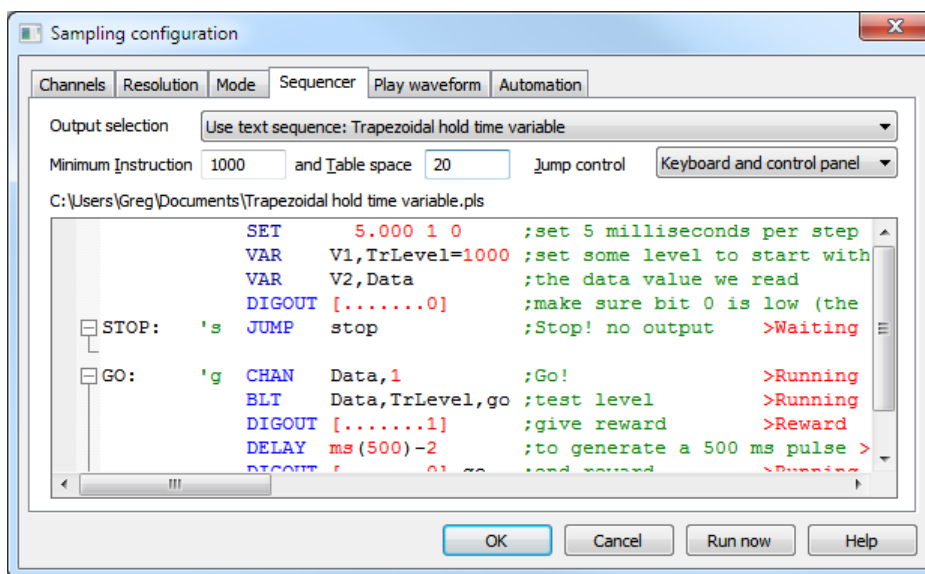
Sequencer

The Sequencer page of the Sampling configuration dialog sets the output sequence to use during sampling. The sequence can either be a *.pls text sequence file stored on disk or it can be a graphical sequence created from this dialog.

The main control is the Output selection drop down list. This enables and disables sequence output and selects an output sequence file or recent sequences generated by the graphical sequence editor. The Select a text sequence file option opens a file dialog to choose the sequence file (*.pls). These files are created with the output sequence text editor or by exporting graphical editor sequences as text.



Output selection options



Minimum Instruction and Table space

Spike2 reserves space in the 1401 for the instructions and table space used by the sequence set in the sampling configuration. However, if you load new sequences during sampling, the replacement sequences may need more space. These two fields reserve extra space; the space set is the larger of that required for the initial sequence and the values you set in these fields. Unless you intend to replace the initial sequence with a text sequence during sampling you should set both these fields to 0. The script equivalent of these fields is SampleSeqCtrl().

Jump control

The sequencer can be commanded to jump to specific places by keyboard commands, the Sequencer control panel and always by a script. You can limit the interactive options to prevent accidental jumps. Choose from:

	Keyboard	Control panel	Script
Keyboard and sequencer control panel	Yes	Yes	Yes
Sequencer control panel	No	Yes	Yes
Script commands only	No	No	Yes

Lower portion of the dialog

If a text sequence file is selected it is displayed in the lower portion of the dialog. To modify an output sequence file you must open it; the easiest way to do this is to double-click in the displayed file. You must close the Sampling Configuration dialog before you can edit the sequence.

If you select the graphical sequence editor, the lower portion of the dialog displays graphical sequencer control settings.

Arbitrary waveform output

You can replay arbitrary waveforms during data capture. Up to ten different “Play wave” areas can be defined for output. Each area is identified by a key code, typically a printing character such as “A”. No two areas may have the same code. You do not have to use a printing character, you can use a two digit hexadecimal code if you prefer, however codes 00, 01 and 02 are not allowed. The format of this code is the same as for marker codes. Because you can trigger waveform output by recording keyboard markers with this code you should make sure that your code usage is compatible with key codes used in the output sequencer.

The waveforms are played from 1401 memory and are copied there just before sampling starts. This reduces the memory available for recording data. The maximum data you can store in the 1401 for replay is the free space in the 1401 less 256 KB which Spike2 reserves for recording. A Micro1401 mk II with 1 MB of memory can store around 700 KB of waveform. A Micro1401 mk II with a 2 MB memory can store around 1700 KB. A Micro1401-3 can store around 3.7 MB. A Power1401 can store from 255 MB to almost 1 GB, depending on the installed memory. Script users can update the memory dynamically during replay, so huge memories are not necessarily required! The maximum size of a waveform area is 32 MB (assuming your 1401 has sufficient memory).

The final sample of the waveform area sets the output level after waveform output ends, so it is usually a good idea to make sure that the waveform output ends with a zero value. Note that the VDACC0–VDACC7 sequencer variables are not updated by arbitrary waveform output.

1401	Micro2	Micro3	Power1,2	Power3
kHz	167	250	250	500

The table shows the maximum rate we measured replaying two waveforms while sampling one. The input and output rates were the same. If you sample or replay more channels or use the output sequencer, the maximum rate will be lower. If your Micro101-3 or Power1401 mk II has the DAC Silo firmware upgrade, the use of the output sequencer or more channels has much less impact on the maximum rate. If your Micro1401-3 does not have the DAC Silo firmware upgrade, the maximum rate is 167 kHz.

Frequency resolution

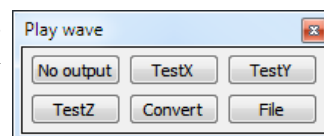
The 1401 DAC output rate is derived by dividing down from a fixed frequency clock. This limits the sample rate resolution to 0.1 microseconds for Micro1401 mk II/-3 and the Power1401 mk II and -3 and 0.2 microseconds (worst case) for the Power1401. These limitations can be significant, particularly if you replay imported data. For example, 44.1 and 48 kHz (often used in .wav files for sound recording) cannot be exactly represented. You can use the `ChanSave()` script command to resample data to a different rate using cubic spline interpolation.

For example, if your 1401 has a sample rate resolution of 0.1 microseconds, and you wanted to achieve 44.1 KHz, the divisor from 10 MHz (the internal clock rate corresponding to 0.1 microseconds) is $10000000/44100$ which is 226.76 to two decimal places. The closest divisor we can get is 227, so instead of 44.1 kHz you would get 44.053 kHz and error of approximately 0.1%.

You should never have any problem matching sampling rates when replaying data that was recorded with your 1401.

Play waveform toolbar

There is a dockable toolbar associated with the waveform output. This is enabled when you are sampling data with waveforms defined. The toolbar can be docked on any edge of the application and can be resized when it is floating. You can assign your own labels to the buttons, or you can let Spike2 generate labels itself of the form *Wave 0*, *Wave 1* and so on. The first button in the toolbar is used to stop a currently playing waveform.



Channels and DACs

Each play wave area contains from 1 to 4 data channels. You can select the 1401 DAC (Digital to Analogue Converter) each channel plays through. The maximum DAC number you can set varies with the type of 1401. All Power1401 models allow DACs 0-7 and all variants of the Micro1401 allow DACs 0-1. It is not an error with a Power1401 to select DACs 4-7 when they are not present; of course, there is no output.

With multiple channels, all channels play at the same rate and all DACs update together. You can play data from waveform and WaveMark channels in a Spike2 data file, or data generated by a script. When data comes from a Spike2 file, the channels need not all have the same sample rate; Spike2 takes the rate of the first channel and interpolates data from the subsequent channels to make the rates the same. Script users can play arbitrarily long data by updating the waveforms in the 1401 online with the `PlayWaveCopy()` command.

The rate at which a wave plays can be modified in the range 4 times slower to 4 times faster than normal (as long as your hardware can output fast/slow enough). From a script, you can change the rate during sampling, even while the wave is playing.

Playing waves

There are several ways to initiate output of a wave during data sampling:

- Click the associated button in the Play waveform toolbar
- Record the associated key code or key set by `PlayWaveKey2$()` in the Keyboard marker channel
- Use the script language `SampleKey()` command to record the key code
- Use the output sequencer `WAVEGO` command

Unless you use the output sequencer to start the output, the associated key is recorded in the Keyboard channel and marks the time at which output was requested.

When a wave starts to play, there is a time delay of one waveform output between the moment that output is requested and the first data point appearing. This is not usually important, but at a slow replay rate, a one sample delay could be significant. This delay can be useful if you are using the output sequencer as it gives the output sequencer the opportunity to know about a change in the DAC outputs *before* it happens.

Triggered output

Each wave you play can be marked as “Triggered”. In triggered mode, when the waveform play is requested, output does not start immediately. Instead, the 1401 hardware waits for a trigger input (high to low edge) on the Trigger input for the Micro1401 and Power1401 unless this is routed to the rear panel Event connector pin 4 (Ground is pins 9-15) by the Edit menu Preferences. In triggered mode, the first data point is output at the time of the trigger.

If you use the output sequencer you choose between triggered and non-triggered in the `WAVEGO` instruction. You can also trigger the wave from the sequencer with the `WAVEST` instruction and detect if the wave has started to play with the `WAVEBR` instruction.

Repeated plays and links

Each wave can be set to play cyclically for a set number of times. You can also link waves together if they have the same DAC output list. Linked waves play at the rate of the first wave, that is the sample rate of subsequent waves is ignored.

For example, you might have a sound output that needs to ramp up, stay at a constant level, then ramp down. This could be done with three waves. The first holds the ramp up waveform, the second which repeats cyclically many times would hold the constant sound, and the final part would ramp down.

Scripts and the output sequencer can command a wave with many cycles to finish the current cycle then continue to the next linked area. You could use this to simulate a blood pressure signal with an occasional errant beat by having two waves, one with a normal beat set to repeat many times and one with the errant beat set to play once. By linking the two areas to each other, you get one errant beat after a fixed number of normal ones. Further, by using the randomization functions in the output sequencer you could produce errant beats randomly and mark the times at which they occurred.

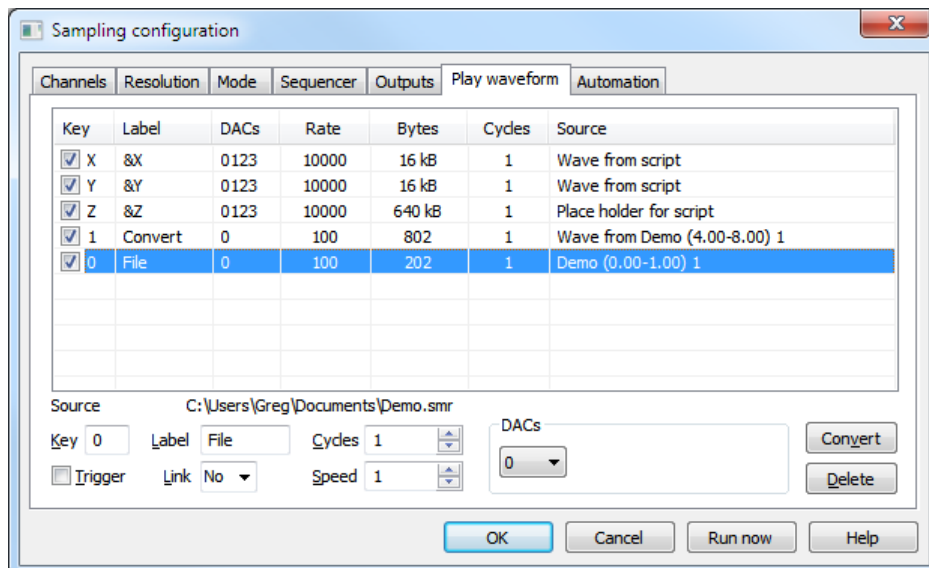
Script-only features

The `PlayWaveCopy()` command allows you to modify the contents of an area while sampling is in progress.

`PlayWavePoints()` allows you to change the size of an area dynamically (but sizes can only be less than or the same as the original size). This lets you allocate an area, then use it to play back waveforms of different lengths. The `PlayWaveKey2$()` command lets you associate a second key with an area. This can be used to play the area, just as the standard key code does, but this code can be changed dynamically, enabling separate key codes for different waveforms played from one area.

Play waveform

The Sampling configuration Play waveform tab lists waves for on-line output. Add waves with the Sample menu Offline waveform output dialog Add to Online button and by scripts. The settings are used the next time you sample. Editable fields for the currently selected wave are at the bottom of the dialog.



Key Wave identifier. A character or two hexadecimal digits in marker code format; you may not repeat a code or use 00. It is added to the Keyboard marker channel on a Play wave toolbar button click.

If a wave is checked in the list, space is reserved for it in the 1401 and if a waveform is defined, it is copied to the 1401 before sampling starts. It can take a noticeable time to copy many megabytes.

Label Up to 7 characters to label buttons in the Play wave toolbar. If you use &, the next letter is underlined and you can use it as a short-cut to the key when the toolbar has the input focus.

DACs A list of the Digital to Analogue Converters to play your waveform out of. You can change the list with the drop down lists inside the group. You cannot set two channels to use the same DAC output.

Rate The number of samples to output per second per channel, set when the wave is added to the list. You can vary the replay rate with the Speed control in the range 0.25 to 4.00. If the speed control is set to any value other than 1.00 the Rate field shows the multiplying factor as well as the rate.

Size This is the number of bytes of 1401 memory needed to hold the wave.

Cycles The times to play the wave. Use 0 for a very large number (about 4 billion).

Source Where the waveform (if supplied) is stored. Below, Name is a data file name, sTime and eTime are the start and end times of the data and chans is the list of channels to read the wave from.

Name (sTime-eTime) chans: a wave in a data file. The full path appears in the Source field.

Wave from Name (sTime-eTime) chans: a wave originally read from a file, held in memory and saved in the sampling configuration. The full path to the file appears in the Source field.

Wave from script: a script generated wave, held in memory, saved in the sampling configuration.

Place holder for script: a script has reserved space. The wave will be generated on line by a script.

Trigger Check this box for waveform output that is enabled by play requests and that starts on the Trigger input. If this is not checked, a play request starts the waveform playing immediately.

Link You can link waves with identical DAC channel lists together. Linked areas play in order with no time gap between them at the rate set by the first wave played.

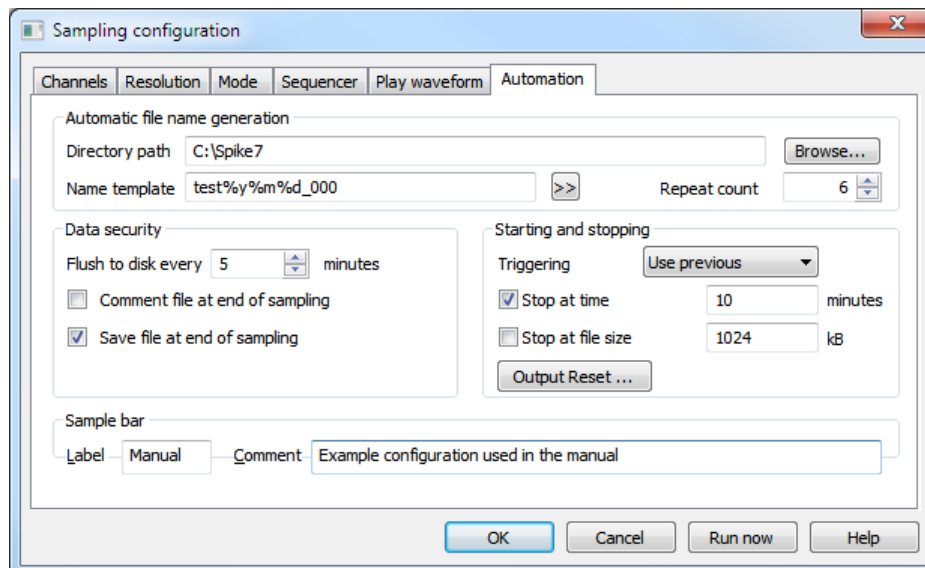
Convert This button changes data held in a file into data held in memory and vice-versa.

Update Applies any changes you have made to the current wave.

Delete Delete the current wave.

Automation

The sampling configuration dialog Automation page sets the file path and name for automatic data filing, sets the security parameters for new data files and restricts the total sampling time and the size of the data file. It also contains the label and comment used when you add the sampling configuration to the Sample bar.



Automatic file name generation

Spike2 samples data to temporary files in the folder set by the Edit menu Preferences. Unless you set a Name template, these files have names like Data1, Data2 with no extension. When sampling ends, you save the file to a name of your choice. You can automate file naming by setting a Name template (up to 23 characters), and Spike2 will generate a sequence of names from it. A blank Name template disables automatic name generation. The file extension is determined by the Output file format set in the Resolution page and should not be set here.

If the template does not end in a number, 000 is added before using it. To make a file name, Spike2 increments the number until a name is formed that is not in use in the Directory path folder (or the current folder if Directory path is blank). A template of test generates test000 to test999. A template of test10 generates test10 to test99. The Directory path must be less than 200 characters long. If the folders set by Directory path and in the Edit menu Preferences option are in the same disk volume, sampled files are renamed rather than copied, which can save a lot of time, especially if you are using automatic sampling on a sequence of files.

With a file name template set, the generated name is used when the data file is saved. A different name can be specified using the File Save As... command. If there are no free names, or the path set for file saving does not exist, you are prompted for a file name.

Date and time specifiers

Within the file name template, the character sequence % followed by one of y, m, d, H, M or S is converted into a two digit representation of the year, month, day, hours, minutes or seconds. %Y is replaced by the year as 4 digits. %D is shorthand for %Y%m%d and %T is shorthand for %H%M%S (we set the digits this way so that sorted names are in time order). If the name ends with a date or time specifier, Spike2 adds _ to the name so that a date or time is not incremented when searching for an unused file name. Of course, as the name does then not end in a number, 000 is also added. For example, if you set the Name template to test%D and you use it on December 19, 2013, the name expands to test131219_000. The >> button to the right of the field lets you insert date and time specifiers at the caret or to replace the selection in the Name template field.

Automatic sampling of a sequence of files

If you set a Name template and a Directory path, check the Save file at end of sampling box and one or both of the Stop at time or file size boxes, you can enable automatic sampling of a sequence of files by setting the Repeat count field. Values of 0 or 1 sample a single file, larger numbers sample a file sequence.

Sampling each file starts and stops based on the conditions in the **Starting and stopping** box. For each file except the last, when sampling stops the file is closed, and all result and XY views created by processing from it are closed. Sampling resumes with the next file in the sequence. The file sequence stops when:

- The number of files set in the **Repeat** count have been sampled
- There is no free disk space
- The name template cannot be incremented or there are no unused names
- There is an error during sampling
- The user clicks on **Stop** or **Abort** in the sampling control panel.

Data security

For efficient sampling and to allow us to retrospectively choose to save data, Spike2 buffers several megabytes of the most recently sampled data in memory and writes data when the buffers are full. The operating system then buffers this data and writes it to disk whenever it chooses. So, if the power failed, unwritten data would be lost. The **Flush to disk every n minutes** field sets how often the data buffered in memory is forced out to the physical disk to guarantee that your data is safe. There is a significant time penalty for doing this (can be seconds) and it interacts with turning write to disk on and off. If you want the fastest possible sample rate you should turn this feature off (by setting a zero period). However, with it on, even if the computer power is lost or the computer crashes, your data should be safe up to at least the last flush time. You must run `S64Fix` (for 64-bit files) or `SONFix` (for 32-bit files) on such a data file to complete the data recovery and to tidy up data blocks written after the last flush.

If the **Comment file at end of sampling** box is checked, you will be prompted to provide a file comment when sampling finishes. This is disabled when sampling a sequence of files as prompting for a comment would interrupt the sequence.

If the **Save file at end of sampling** box is checked, the new data file is saved to disk automatically when sampling finishes. If automatic file name generation is in use, the generated file name is used, otherwise the usual prompts for a file name are provided. This box must be checked if you want to sample a sequence of files automatically.

Starting and Stopping

The **Triggering** field allows you to preset the state of the sampling control panel **Trigger** check box:

- Use previous** Uses the current state of the **Trigger** check box
- Not Triggered** Forces sampling to start immediately; clears the **Trigger** check box
- Triggered** The first sampled file is triggered, any repeats are untriggered
- All triggered** The first file and all repeats are triggered

You can cause sampling to stop automatically at a set run time, or when the data file is a set size. If you do not check a box, the associated limit is not used. In general, you will get a little more data than implied by an active time or data limit as data buffered in the 1401 at the time the limit was reached will be added to the file. To sample a sequence of files automatically you must check at least one of **Stop at time** or **Stop at file size**.

Output Reset...

You can choose to have the DAC and digital outputs set to known values when a sampling configuration is loaded, and more usefully just before sampling starts and after sampling ends. Click this button to open the **Output Reset** dialog. Values set from here are stored in the sampling configuration and override application settings set from the **Edit** menu **Preferences** in the **Sampling** tab.

Sample Bar

The **Automation** page also holds a label of up to 8 characters and a comment of up to 80 characters for the **Sample bar**. When configurations are saved to a `.s2cx` or `.s2c` file, they can be added to the **Sample bar** by the **Sample** menu **Sample Bar List...** command and any label or comment is used to provide information about the configuration.

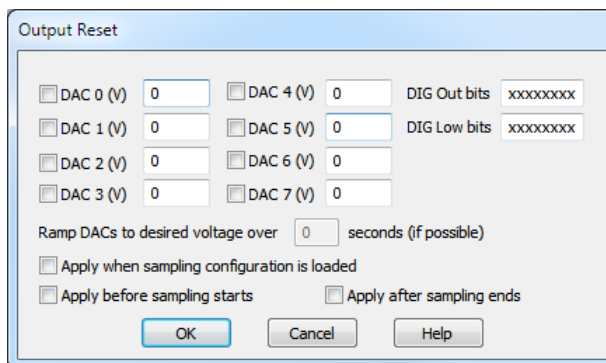
If you include an ampersand (&) in the label, the following character can be used with the **Alt** key as a shortcut to load and run the configuration. See the **Script bar** documentation for details and caveats.

Once a configuration is in the **Sample bar**, you can open a new data file with a mouse click or **Alt+key**.

Output Reset

The normal reset state of a 1401 is all DAC outputs at 0 Volts and all digital outputs set low. Spike2 leaves all 1401 outputs alone except when sampling. However, in some situations it can be important that the 1401 outputs are returned to a known state after sampling in case sampling was interrupted; this is where you would use the Output Reset dialog.

There are two sets of Output Reset information. One is associated with the Spike2 application, the other with the sampling configuration. The effect of the two sets of information is as if the applications settings were applied followed by the sampling configuration settings.



Feature	Application	Sampling Configuration
Opened from	Edit menu Preferences Sampling tab	Sampling configuration Automation tab
Dialog title	Application Output Reset	Output Reset
Where is data saved	In the registry	In the sampling configuration
Apply on load	When the program starts	When sampling configurations load
Priority	Values may be over-riden	Values are always used
Use for	State you always need applied	State dependent on the sampling configuration

The image shows the dialog when opened from the Sampling Configuration Automation tab. When opened from the Edit menu Preferences, Sampling tab the dialog title changes to Application Output Reset and the Apply when sampling configuration is loaded check box becomes Apply when application starts.

DAC values

Spike2 supports up to 8 DACs per 1401 interface. You can set the reset value (in Volts at the 1401 output) for each DAC. Only DACs that are checked (and exist in your interface) will be reset. The normal reset level is 0 Volts, but you can set any value within the range of the DACs.

Dig Out bits

This field sets the reset values for bits 15-8 (from left to right) of the digital outputs as 8 characters. Each character can be 0, 1 or x standing for low, high and no change.

Dig Low bits

This field sets the reset value for bits 7-0 (from left to right) of the digital outputs as 8 characters. Each character can be 0, 1 or x standing for low, high and no change.

Ramp DACs to desired voltage over ... seconds (if possible)

Currently this field is inoperative and the DACs change to the reset value in one step. When implemented, the DACs will ramp from their current value to the reset value over the time period specified.

Apply...

If you do not check any of the Apply... boxes, this dialog has no effect. You can choose when to apply the reset state: when the Output Reset information is loaded, before sampling starts, after sampling ends, or at any combination of these times.

Data buffering

When sampling, Spike2 keeps several megabytes of the most recently sampled data in memory. Having buffered data allows you to run with "write to disk" disabled, yet still allowing you to see recent data and giving you the possibility of retrospectively deciding if data is worth saving or not. How this is done is different between the old 32-bit files and the newer 64-bit files. In both cases, buffer space is allocated to

channels in proportion to the expected data rates for each channel (as set, by you, in the sampling configuration). For waveform channels, the expected rate is the sample rate multiplied by the size of each sample. For Event, Marker and extended Marker data (TextMark, RealMark, WaveMark), the expected data rate is the **Maximum event rate** field multiplied by the size of each item. It is important that you set this field to a realistic peak event rate over say 10 seconds; setting the peak instantaneous rate will result in inefficient allocation of resources.

Having buffers allows us to allow retrospective changes to what data should be saved. Changes are additive; that is you can have saving logically disabled then mark regions for saving with `SampleWrite()` or triggered or timed sampling.

32-bit (.smr) file buffering

Buffer space is allocated in terms of complete disk blocks, as written to the data file. These blocks are typically a multiple of 4096 bytes up to a maximum size of 32768 bytes. Each channel has a list of buffers it can use, and these are filled in sequence. Each buffer can be marked for saving or not saving. When all buffers become full and more data arrives, the oldest block is written to disk if it is marked for saving, then that block is used for new data. If any data in a block is marked for saving, the entire block is saved. If you use the `Modified(0,x)` command or set automatic flush to disk, and it causes any data to be saved, any older blocks that were marked as not saving are emptied.

64-bit (.smrx) file buffering

During sampling, each channel is allocated a circular buffer to hold the most recent data. The channel also has a save/no save state and list of times at which the save/no save state changes. When the circular buffer becomes full, the oldest data is ejected from the buffer. If it is not marked for saving, it is lost. If it is marked for saving, the data is added to the channel write buffer (a 64 kB buffer that is an image of a disk block for the channel). When the channel write buffer becomes full, it is written to the disk. If you use the `Modified(0,x)` command or set automatic flush to disk, it has the effect of forcing all data marked for saving to the channel write buffer (which causes it to be written if it becomes full) and then if the channel write buffer is partially full, the partial buffer is also written. Times before the last saved data time are removed from the save/no save list (as once data is committed to the channel write buffer you can no longer change the save/no save state for earlier data). However, any data in the circular buffer that is not marked for saving is preserved.

Flushing data to disk

On modern operating systems, when an application writes data to a disk file, this will not usually write data to the disk surface. For performance reasons, data is written into a complex buffering system which is designed to keep the entire system responsive. This means that should the application crash, the operating system will close down open files and the files will be intact. However, if the system fails (due to power loss or a system crash), all the buffered data in the system does not get written, resulting in significant data loss.

To work around this, Spike2 supports an automatic flush to disk option or you can use the `Modified(0,1)` command to request that the file data is transferred to the physical disk.

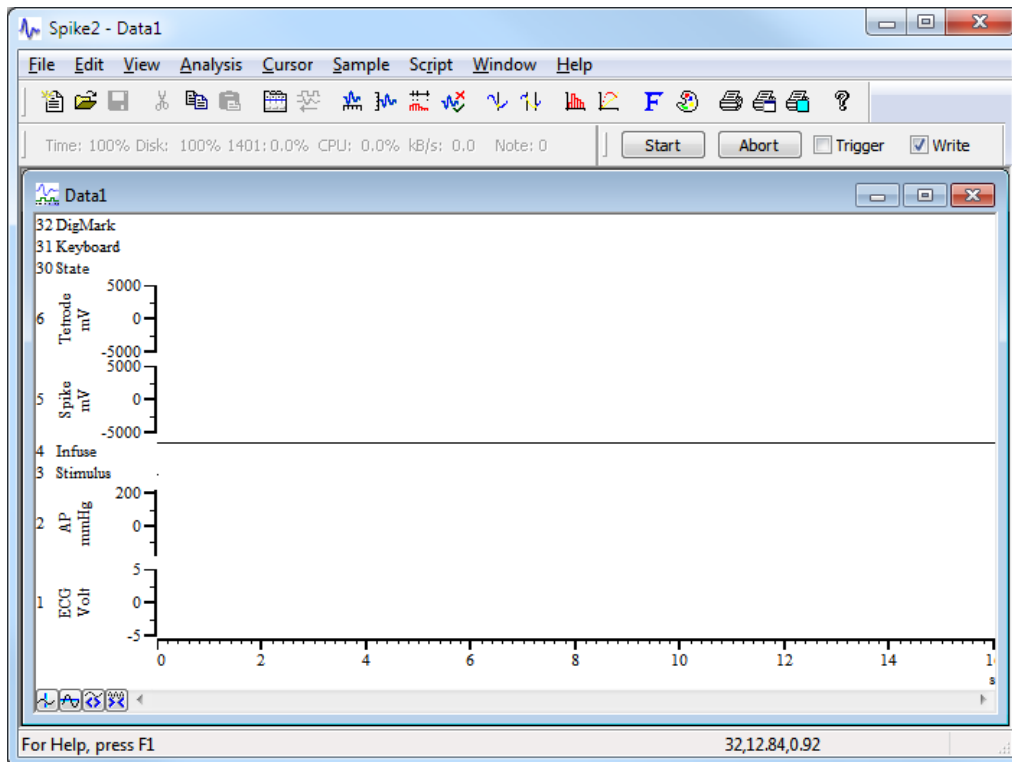
This used to work well, but more modern disk systems now include huge memory buffers internally. It turns out that if you have a "primitive" disk system, when you ask the operating system to flush the buffers for a file through to the disk, this works very quickly as the system "knows" which buffers belong to the file and need to be written to which physical disk sectors. Unfortunately, when the disk itself buffers vast quantities of data, the disk has no idea which buffers refer to which files. The only way to guarantee that the data for a file is on disk is command the disk to write ALL buffered data. We had one case of a user who upgraded to a new, vastly faster computer and found that the first disk flush took 15 seconds when their previous, ancient machine did it instantly.

If you hit a problem like this, one way around it is to make sure that your disk has an interruptible power supply and then use `Modified(0,0)` at suitable moments to get data through to the operating system without triggering the flush to disk. Even if your system should fail, the disk should be able to complete the physical writes.

Opening a new document

Once you have set the sampling configuration you can open a new data document. Select **New** from the File menu, then **Data Document** for the file type. Data documents differ from all other Spike2 documents as they are always stored on disk. Other document types are kept in memory until you save them. We keep data

documents on disk because they can be very large. When you save a new data document after sampling, Spike2 moves it to the disk volume and directory you specify. When you use the File menu New command, Spike2 creates a temporary file in the directory specified in the Edit menu Preferences. If you do not specify a directory in the preferences, the location of the temporary file is system dependent.



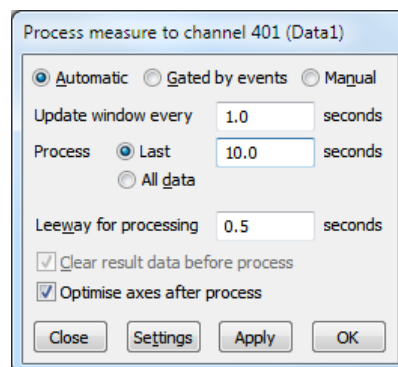
The exact appearance varies, depending on the configuration. Sampling begins when you click Start in the Sample control toolbar (the Sample menu duplicates the controls in this window). If the Trigger box is checked, sampling waits for an external signal. You can set the display and analyses required before sampling. For example, to set an interval histogram you can select that analysis exactly as you did in the *Getting started* chapter. There is a difference, however. When you click on New, a new dialog appears.

Process dialog for a new file

You create result views with the Analysis menu New Result View command in the same way as when working off-line. However, the Process dialog, which controls when and how to update the result window, has additional options to work with a data file that grows in length. The radio buttons select Automatic, Gated by events or Manual updates (see the *Analysis menu* for a full description).

The standard on-line mode is Automatic. This mode adds new data to the result at a user-defined interval. You can choose to accumulate a result for all the data, or produce a result for the last few seconds. The other two modes are for specialised uses and should not be used unless you are certain they are what you want.

This dialog disappears once you select either OK or Cancel, however you can recall it with the Process command in the Analysis menu.

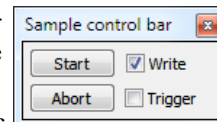


Sample control toolbar

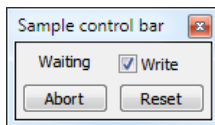
The Sample control toolbar holds several buttons and a check box and controls the data sampling process. You can dock this bar to any edge of the Spike2 application window or leave it floating. The toolbar becomes visible (if it was invisible) whenever sampling starts unless the start command comes from the script language or the Edit Preference to prevent this is set. The position of the toolbar is saved in the sampling configuration.

However, if it is visible and docked, we do not reposition it as we assume it is where the user wants it.

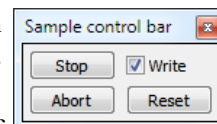
The **Trigger** field controls whether sampling starts immediately when you click **Start** or if it waits for a low-going TTL compatible trigger pulse. The **Triggering** field in the **Automation** tab of the sampling configuration dialog sets the initial **Trigger** state.



If you click **Start** with **Trigger** checked, **Waiting** flashes in place of the **Start** button until a suitable signal is applied to the **Trigger** input. Use this method to synchronise the start of sampling with an external event. Sampling starts within 1 or 2 microseconds of the external signal.



You can also decide which portions of your data are to be saved on disk, and which portions are of only transitory interest. The controls are duplicated in the **Sample** menu; however the **Sample** control toolbar needs fewer mouse clicks. The buttons are:



- Start** This is displayed before data capture starts. Click the button to start sampling. If **Trigger** is checked, Spike2 waits for the trigger before continuing.
- Stop** This is displayed while data is sampled. Click this button to stop sampling and keep the data. If no data was saved, the empty file is discarded.
- Abort** This button is used to abandon sampling and discard the new file. You can use this button before sampling starts, or while sampling is in progress.
- Reset** This button appears when you click **Start**. It stops sampling, discards any saved data, and waits for you to start sampling again with the same document.

Write check box and Not saving

The **Write** check box is normally checked, to save data to the data document. If you clear this box, no data is saved until it is checked again. If you are running in **Triggered** or **Timed** sample modes, clearing this check box will also disable writing for these modes (even if a trigger or time interval occurs).

Spike2 keeps a certain amount of data in buffers in memory, so it can display recent data on screen even if you have decided not to save it to disk. This saved data is used in **triggered** mode to save pre-trigger data and can also be marked for saving with the `SampleWrite()` script command.

You can write enable and disable individual channels from the `SampleWrite()` script language command. This command can also mark data regions for saving when the channel save state is set to not saving. If you use this command to set a state where some channels are enabled and some are disabled, the check box is drawn in an intermediate state (neither checked nor clear).

Whenever you change the state of the check box or use the script command to enable or disable all channels, a marker is added to the keyboard channel (code 00 when writing is enabled and code 01 when it is disabled).

Whenever any channels are not saving, the **Write** text moves from side to side to alert you and new data for those channels (apart from **WaveMark** data and **Sonograms**) draws in the **Not saving to disk** colour as a warning that data is not being saved.

High sampling rates

The code that transfers sampled data to disk runs in a separate, high priority thread. This ensures that data is saved unless another program creates a higher priority process that uses all the computer time. Buffer overflow can occur if the data rate is so high that the 1401 device driver cannot empty 1401 memory before it has become full. This should not occur with USB2 interfaces unless the overall data rate is very high or there are bandwidth bottlenecks in the disk system. Spike2 detects buffer overflow and stops sampling if this happens.

If you suffer from buffer overflow problems, please check the following:

- Check that you have the latest 1401 firmware. The Help menu **About Spike2** dialog box will tell you if more recent firmware is available. Spike2 will refuse to sample if your firmware is seriously out of date.
- If you use USB to connect your 1401 and your 1401 has a USB2 interface, make sure it is connected to a USB2 port on your computer. Using a USB1 port (some older computers have both USB1 and USB2 ports) will reduce the maximum transfer rate by a factor between 5 and 40 depending on the type of 1401.
- You can get a measure of the fastest data transfer rates available for your combination of 1401, interface and computer by using the **Try1401** program. Select **Show estimated transfer speeds** in the **Tests** menu

Settings command and then run the DMA test.

If the 1401 detects that the host computer is slow writing to disk, it requests a “catch-up” mode where Spike2 abandons on-line display of new data (the display for new data is greyed out) to avoid competition between writes and reads in the disk system.

If the 1401 detects that the input event rate is too high for the 1401 to process, the special keyboard marker code FF is added to the keyboard channel. Sampling does not stop, as subsequent event times will be correct. If this happens, check that you have set realistic expected sample rates for the event channels.

The Sample Status bar

The Sample Status bar gives you an indication of how hard the 1401 and the host PC are working to capture and transfer data to disk. The bar also displays how much sampling time and disk space remains before sampling will stop. If the bar is not visible during sampling you can show it from the Sample menu or by right-clicking in the toolbar area and selecting **Sample Status**. The bar is normally made visible when a data file is opened for sampling unless the data file was created by a script or an Edit menu Preferences option is set to prevent it opening. The bar shows five values:

- Time** Percentage of sampling time remaining before sampling stops. You can limit the sampling time in the Automation tab of the Sampling Configuration. If you do not limit the time, you can run for up to 8×10^{18} clock ticks when sampling to a 64-bit file or for some 2 billion clock ticks when sampling to a 32-bit file (you can see how long this is in years, days, hours minutes and seconds in the Resolution tab of the Sampling configuration).
- Disk** Percentage of disk space remaining before sampling stops. You can limit the disk space in the Automation tab of the Sampling Configuration. If you do not limit the disk space, for 32-bit files the limit is set to 1 TB in big file mode or 2 GB if not in big file mode and no limit for 64-bit files. The free space on the disk is checked just before sampling starts and the limits are reduced if there is less free space than the limit. Sampling will stop if you get within a few megabytes of the free disk space when you started. If other activities are also filling the disk sampling may stop sooner, and possibly in a less controlled manner.
- 1401** The percentage of the available time that the 1401 interface is using to transfer captured data back to the host computer and match any spikes to templates. This should be fairly constant with waveform channels, but will increase when the rates of event or WaveMark channels increases. It can also be increased by a slow host interface or problems with the host computer disk system. If it heads towards 100% you are in trouble.
- CPU** The percentage of the available time that the data capture thread in the computer is using to process the incoming data from the 1401 and write it to the data file. If you have a computer with multiple processors, this is the proportion of the time on a single processor. This will usually be less than 10%.
- kB/s** The number of kB (kilobytes) of sampled data that is being transferred to the host per second.
- Note** The number of warning messages written to the Sampling Notes log. Double click this field to open the **Sampling Notes** window during sampling.

The 1401, CPU and kB/s fields display as n/a if the 1401 firmware is not sufficiently up to date, but this should never happen in Spike2 version 8 as we refuse to sample if old firmware is detected. You can download firmware updates from the CED web site: Micro1401-3, Micro1401 mk II, Power1401, Power1401 mk II, Power1401-3.

Saving configurations

It would be very tedious if you had to setup the exact screen configuration you wanted each time you sampled data. To avoid this, you can save and load sampling configurations from the File menu. The saved sampling configuration includes:

- The position and size of the application window
- The list of channels set for sampling and their sampling parameters

- The position of all windows associated with the new file
- The displayed channels and event display modes of the channels in time windows
- The name of any output sequence document to be used during sampling
- The list of waves and any associated waveform data for on-line waveform output
- The processing and update modes and positions of all result windows

The configuration does not include the contents of result windows. Whenever sampling finishes, the application saves the configuration as `last.s2c`. When you run Spike2, it searches for and loads the configuration file `default.s2c`. If this cannot be found, it uses `last.s2c`. These files are kept in the directory from which Spike2 was run. Remember that you can always recall the configuration that you used most recently, even if you forgot to save it.

If you have several configurations that you use very regularly, you can add them to the Sample bar with the **Sample** menu **Sample Bar List...** command. Once you have done this you will be able to start sampling with a saved configuration by clicking a button on the Sample bar (see the *Sample menu* chapter for a full description of the Sample bar). Alternatively, you could keep shortcuts to configuration files on your desktop and start Spike2 by double clicking them.

Warning

We suggest that you do not rely on `last.s2c` for important sampling configurations as it is overwritten each time you sample. It is better to save your configuration to a named file and load it using the **File** menu or the **Sample Bar** or by double clicking the file.

Sequence of operations to set the configuration

This section describes a sequence of operations that build a new sampling configuration from scratch. You will find that once you have built a few configurations, it is much simpler to load an existing configuration and change the sections that do not fit your requirements, rather than re-build entirely. The steps are:

1. Open the **Sampling Configuration** dialog from the **Sample** menu.
2. In the **Channels** tab set the channels to be sampled and their sampling rates.
3. Set the **Resolution** values to give the best fit of run time and sampling rates for waveform channels.
4. Select the appropriate **Sampling Mode** for your needs.
5. In the **Sequencer** tab select an output sequence file or create a graphical sequence or select **None**.
6. Set the **Automation** values as required by your application.
7. In the **Play waveform** tab select any waves needed for waveform output.
8. Click the **Run now** button.
9. Arrange the time view as you require and add any duplicate windows.
10. Add required Analysis processes generate result windows, set their update mode and screen position.
11. Add Measurement processes to generate XY views or send their results back to the original time view.
12. Use the **File** menu **Save Configuration** command to save the configuration.

Once you have saved a configuration, you can re-use it by loading it before you use the **File** menu **New** command to start a new data file. You can combine the loading and opening a new file by adding the saved configuration to the Sample bar with the **Sample** menu **Sample Bar List...** command; this lets you open a new file (and even start sampling) with a single mouse click or **Alt** key combination in the Sample Bar.

5: Output sequencer

Output sequencer

While sampling data you can generate precisely timed digital pulses and analogue voltages, monitor your experiment and respond to input data in real time with the Spike2 output sequencer.

Overview

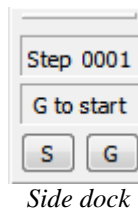
An output sequence is a list of up to 8191 instructions. It runs at a constant, user-defined rate of up to 100 instructions per millisecond. The sequencer has the following features:

- It controls digital output bits 15-8 with changes on the next clock tick and bits 7-0 as unlocked changes.
- It controls the 1401 DACs (Digital to Analogue Converters) to produce voltage pulses and ramps.
- It can play cosine waves at variable speed and amplitude through the DACs.
- It can test digital input bits 7-0 and branch on the result.
- It can record the digital input state or an 8-bit code to the digital marker channel.
- It supports loops and branches and can randomise delays and stimuli.
- It has 256 variables (v1 to v256) that can be read and set by on-line scripts.
- It supports a user-defined table of values for fast information transfer from a script.
- It can read the latest value from a waveform channel and the number of events from an event channel. Using this information, real-time (fractions of a millisecond) responses to input data changes are possible.
- It can control and monitor the arbitrary waveform output.

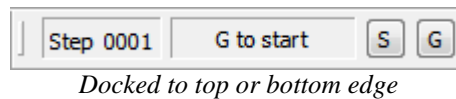
You write sequences with the text editor where each line of text generates one instruction or with the graphical editor where each graphical item generates one or more instructions.

Sequencer control panel

You show and hide the control panel with the **Sequencer Controls** option in the **Sample** menu or by right clicking on any toolbar to activate the context menu. You can also dock it on any edge of the Spike2 application window. When undocked, the control panel displays sequence entry points as the key that activates them and a descriptive comment. When docked, the keys display as buttons and the comment is hidden; move the mouse pointer over a key to see the comment as pop-up text. Click the mouse on a key and the sequencer will jump to the instruction associated with the key. The key is also stored as a keyboard marker. This is equivalent to pressing the same key in the time window or using the script language `SampleKey()` routine.

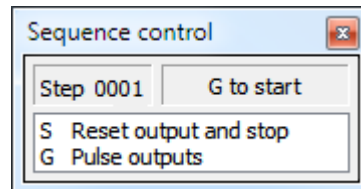


The control panel displays the next sequence step and any display string associated with it. You can use display strings to prompt the user for an action or to tell the user what the sequence is doing. Display strings are set with the text editor. The sequencer always starts at the first instruction. You cannot re-route the sequencer until sampling has started.



Docked to top or bottom edge

The control panel is usually displayed if you start sampling from a Spike2 menu command. If the sample command comes from the script language or an Edit Preference is set to prevent it opening, the control panel visible state does not change. The control panel position is saved in the sampling configuration. However, the position is restored only if the control is currently invisible or floating, and the saved position was floating; if the control panel is docked, we assume it is positioned where you want it. When floating, you can resize the control to display longer comments.



Not docked (floating)

Sequence jump disable

Sometimes you may want to stop users activating sequence sections with the keyboard or from the control panel. The **Sequencer jumps controlled by field** in the **Sequencer** tab of the **Sampling configuration** dialog lets you do this. The script language `SampleKey()` command can always activate sequencer sections.

Creating sequences

There are two ways to create output sequences: as an output sequence text file with the `.pls` extension or as part of the sampling configuration using the graphical sequence editor. The table summarises the main differences between them.

	Text sequence	Graphical sequence
Edited with	Built-in text editor	Built-in graphical editor
Visualise output	No	Yes
Stored as	Output sequence <code>.pls</code> text files	Part of the sampling configuration
Implemented by	Machine code like language	Drag and drop editing
Ease of use	Takes time to learn	Very easy to learn and use
Flexibility	All features available	Uses pre-set building blocks
Timing	One instruction per text line	Several instructions per item

Graphical sequences are much easier to generate than text sequences. However, they have limitations and you can write more complex sequences using all sequencer features with the text editor. Sequences produced by the graphical editor are converted internally to the `.pls` file format before they are used. You can save a graphical sequence as a `.pls` file; you cannot convert a `.pls` file into a graphical sequence.

The rest of this chapter describes the graphical output sequence editor, then the text editor, and finally the low level instructions used by both. You do not need to read about the low level instructions to use the graphical editor. However, knowledge of how the sequencer works will give you a better understanding of its capabilities and limitations.

Sequencer speed

The output sequencer runs at a rate set by a clock inside the 1401. You set the clock rate in milliseconds per tick in the Sequencer Tab of the Sampling configuration dialog when using the graphical editor or using the `SET` or `SCLK` directives in the text editor. The script command `SampleSeqClock()` can also change the rate. Supported 1401s allow intervals from 0.01 up to 3000 ms (older 1401s had a lower limit of 0.05 ms). If you set intervals less than a millisecond or use the sequencer text editor you should read the following information.

Sequencer technical information

The sequencer clock starts within a microsecond of recording time zero and is time locked to the 1401 event timing and waveform channel recording. Each clock tick books an interrupt to run the next sequencer instruction and updates digital output bits 15-8 if they were changed by the previous instruction.

An interrupt is a request to the 1401 processor to stop what it is doing at the earliest opportunity and do something else, then continue the original task. The time delay between the interrupt request and the instruction running depends on what the 1401 is doing when the clock ticks and the speed of the 1401. This delay is typically a few microseconds, so instructions do not occur precisely at the clock ticks but changes to digital output bits 15-8 do. Changes made by the sequencer to the 1401 DACs and digital output bits 7-0 occur a few microseconds after the clock tick.

The table shows the minimum clock interval, the timing resolution, the approximate time per step, the extra time used for cosine and ramp output and the time penalty for using the slow `DIV` and `RECIP` instructions for the Power1 and the Micro2 (more recent devices will be quicker). Notice that the first two are in units of milliseconds and remainder are in microseconds.

	Power1401	Micro1401
Minimum tick (milliseconds)	.010	.010
Resolution (milliseconds)	.001	.001
Time used per tick (microseconds)	<1	~1
Cosine penalty/tick (microseconds)	0.55	~1
Ramp penalty/tick (microseconds)	0.5	0.7
<code>DIV</code> , <code>RECIP</code> penalty (microseconds)	<1	<3

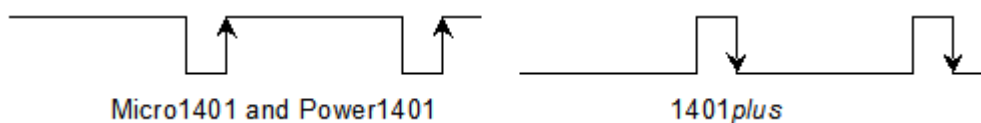
The Minimum tick is the shortest interval we allow you to set. The Time used per tick is how long it takes to process a typical instruction. The Cosine penalty/tick is the extra time taken per cosine output. The Ramp penalty/tick is the extra time taken per ramped DAC. Time used by the sequencer is time that is not available for sampling, spike sorting or arbitrary waveform output. To make best use of the capabilities of your 1401 you should set the slowest sequencer step rate that is fast enough for your purposes.

If you overload the system so much that the output sequencer cannot keep up, sampling stops with an explanatory message.

The interval you set must be a multiple of the Resolution field for your 1401. This is 0.001 ms for the Power1401 and Micro1401. Spike2 will not use a sequence if the interval is not an exact multiple of an achievable resolution for your 1401 as this would lead to inaccurate timing.

Sequencer clock output

The output of the clock that controls the sequencer is available on the 1401 front panel Clock connector. This TTL output signal starts high and the sequencer steps are synchronised with the rising edges of this output. The minimum pulse width is 1 microsecond. If you are upgrading from a 1401plus (not supported for sampling in Spike2 version 8), note that this had the inverse output.



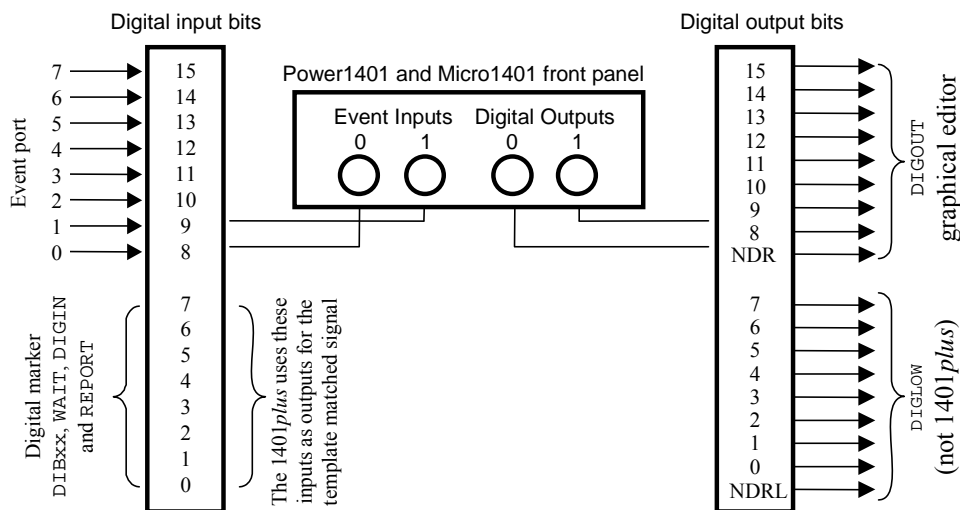
This clock runs whenever you sample with an output sequence. It can be used to synchronise external equipment. For example, if you connect this output to a counter, and place the counter in the field of view of a video camera that is used to record some other aspect of your experiment, the number on the counter links visual data with an exact time in the Spike2 file. If your sequencer ran at 1 millisecond per step, the counter would display time in milliseconds.

Digital input and output

The 1401 family has 16 digital inputs and 16 digital outputs. Digital input bits 15-8 are the event ports and are not used by the output sequencer. Digital input bits 7-0 are used for the Digital marker channel; you can also test these bits from the sequencer.

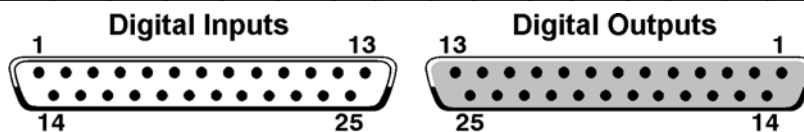
The sequencer controls digital output bits 15-8 individually to generate accurately timed pulses. Output bits 7-0 can be set with the DIGLOW instruction. The on-line template matching code can also use bits 7-0 of the output to signal spikes that match templates.

In a text-based sequence, the digital outputs are controlled by the DIGOUT and DIGLOW instructions and the digital inputs are tested with DIBxx, WAIT, DIGIN and REPORT. The graphical editor controls output bits 15-8 and tests input bits 7-0 with the delay and branching instructions.



The digital input and output ports are 25-way connectors. The data and ground pins are the same on both. See your 1401 Owners Manual for full details of all pins.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	GND
Pin	1	14	2	15	3	16	4	17	5	18	6	19	7	20	8	21	13



Digital output bits 8 and 9 are connected to the front panel as Digital Outputs 0 and 1 in addition to the rear panel. If you have the Spike2 top box, the remaining digital output bits 10-15 are also available on the front panel as Digital Outputs 2 to 7. The NDR output (New Data Ready, output connector pin 12) pulses for 1 microsecond after a change to the digital output bits 8 to 15. NDRL (output connector pin 23) pulses for 1 microsecond after a change to output bits 0 to 7. The pulses are low-going for a Power1401mk 1 and high-going for all other 1401s.

The graphical editor

With the graphical editor, you can generate complex sequences without any knowledge of the underlying control language as used in the output sequencer text editor. There are limits on what can be achieved with the graphical editor. For example, the graphical editor does not have access to tables of values. In complex cases you can use the graphical editor to generate most of the code, then use the text editor for the final touches.

Structure of a graphical sequence

A graphical sequence is defined in Sections. Each section has a duration, a repeat count, an optional key that can jump control to the section and an action to take when the end of the section (and any repeats) are completed (stop, or jump to another section). There is a list of up to 27 sections that you can define: the Initial section (which runs when the sequence starts and may be the only one you need) plus 26 more called Section A, Section B, and so on to Section Z. Most graphical sequences use only a few sections.

Structure of a Section

Most sections are quite simple and contain timed digital output and DAC changes. If you assign a key to a section, the section can be started by a user keypress or by clicking the appropriate button in the sequencer control panel. Most sections run linearly from start to finish executing the instructions that you have configured in time order. You can make a section non-linear in two ways:

1. By making the section wait for a condition (during which period the section time base is effectively paused)
2. By making the section branch to a label within the section, or to the start of another section

Getting started

If you are new to the graphical editor, start by following the Getting started tutorial, which will introduce you to graphical editing. You can then work through the more detailed reference that follows, starting with a description of the Graphical editor Tab and then the section on Graphical editing and the Graphical palette.

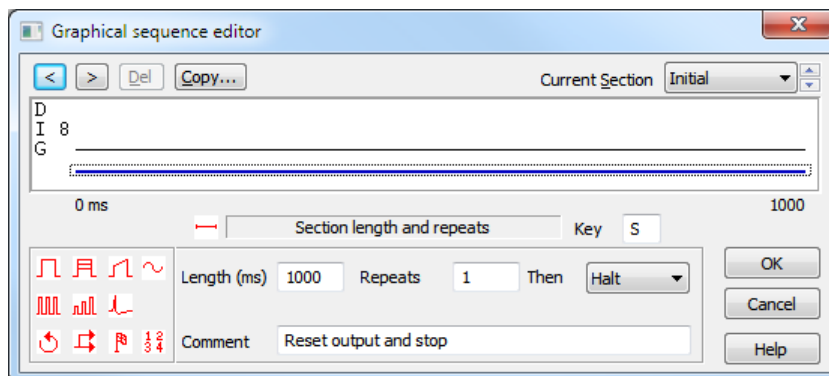
Getting started

To get accustomed to the graphical editor, we will produce 10 millisecond wide TTL pulses at 1-second intervals from digital output bit 8. We will associate the key G for Go with the pulses, and the key S for Stop will stop them.

Open the Sampling configuration dialog Sequencer tab and select Use graphical editor. Click the Clear graphical editor button to remove any previous sequence. Set the Sequencer time resolution to 10 milliseconds and check the Show time as milliseconds box. We do not need any DAC outputs, so clear all the DAC outputs check boxes. We only need one digital output, so check the digital output 8 box and clear all the others. Now click the Graphical editor button.

At the top right of the new dialog there is a drop down list to select the current sequence section. A section has a length and repeat count. When the repeats are done, the section either stops or jumps to another section. To start with, make sure Initial is selected as the Current section. The Initial section always runs first and it sets the initial conditions.

The graphical representation of a section always contains a *control track* drawn as a thick line at the bottom. We choose output on digital bit 8 only, so there is a single digital output in the remainder of the space. There is always one item selected in this area; the selected item has a grey rectangle around it. Click on the control track now.

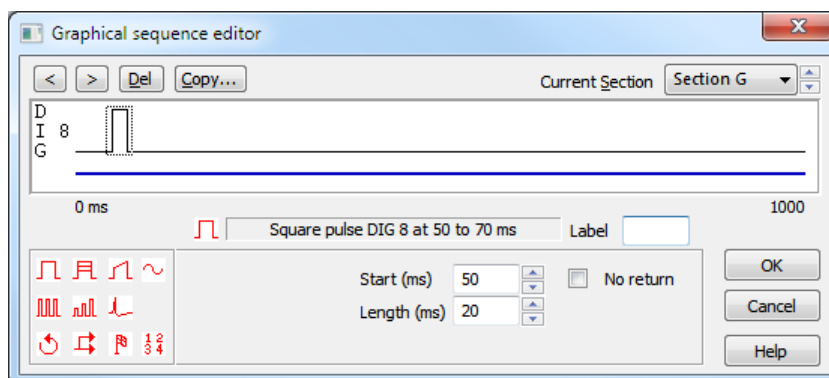


In addition to setting the initial state, we will use this section to stop our pulse output, so set the **Key** field to **S**, and set the **Comment** field to **Reset output and stop**. This comment will appear in the sequencer control panel for the **S** key. You can leave the **Length**, **Repeats** and **Then** fields at their default values (1000, 1 and Halt).

The next task is to create the pulse train and associate it with the **G** key. Select **Section G** in the **Current section** field (or any other section apart from Initial). The control track will be selected; click on it if it is not selected. Set the **Key** field to **G**. Set **Repeats** to 0 to mean repeat forever and the section length to 1000 milliseconds. The **Then** field will grey out as the repeats never end. Set the comment to **Pulse outputs**. There is no intrinsic link between section **G** and the **G** key; however, it is often convenient to use sections with the same identifier as the key used to trigger them.

There is a palette at the bottom left of the dialog; you can drag icons from the palette and drop them on the DAC, digital output and control tracks. Click on the top left item in the palette and drag it to the digital output track and release it to create a pulse. Edit the start time of the pulse to 50 and the length to 10 milliseconds.

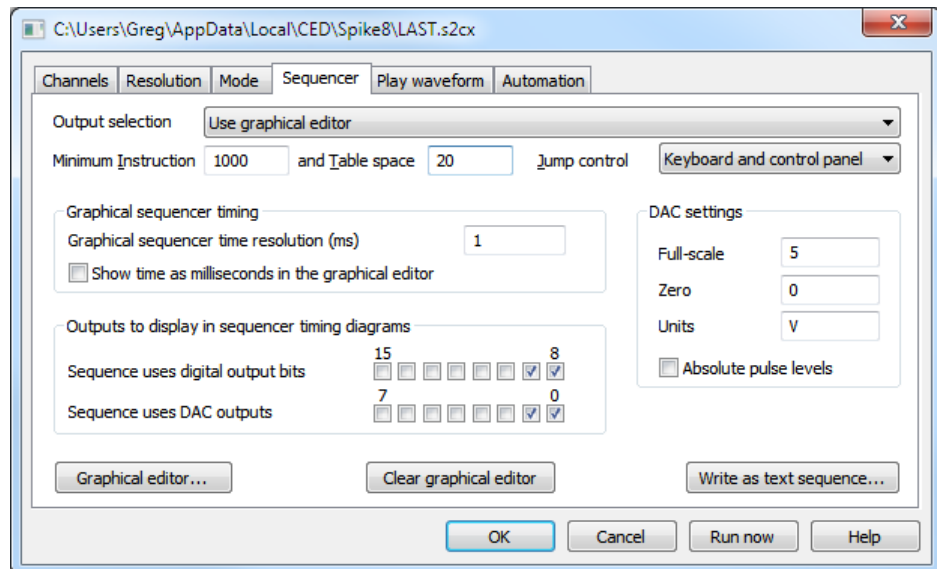
This completes the pulse setup. Click the **OK** button to return to the Sampling configuration dialog. Click the **Run now** button to start sampling with the output sequence you have just created. The sequencer control panel will display two items that you can select: **G Pulse outputs** and **S Reset output and stop**.



Start sampling – the sequencer will run the **Initial** section, which sets the starting values for all the digital and DAC outputs we have chosen to control. We did not request any other action in this section so nothing will happen until you use the **G** button in the sequencer control panel or select the sampling window and press the **G** key to start the output pulses from digital output 8. You can stop the pulses with the **S** button in the control panel or the **S** key on the keyboard.

Graphical editor tab

To view the graphical editor, open the Sampling configuration dialog and select the **Sequencer** tab. Then select **Use graphical editor** from the **Output selection** drop down list. The editable fields in this dialog set values that apply to the entire sequence:



Sequencer jumps controlled by

This field applies to graphical and text sequences. It allows you to stop interactive control of sequence jumps to prevent accidental changes caused by a user keypress or mouse click in the sequence control panel. The `SampleKey()` script command can always cause the sequence to jump. You can choose from: **Keyboard and sequencer control panel**, **Sequencer control panel** and **Script commands only**. The script language equivalent is `SampleSeqCtrl()`.

Sequencer time resolution

This sets the time resolution of your sequence and the clock interval of the sequencer clock. This is also the minimum duration of any pulse. All actions in the sequence occur at integer multiples of the time you set here. You can set values in the range 0.01 to 3000 milliseconds. You need a Power1401 or Micro1401 mk II or -3 to set values less than 0.05 milliseconds. Some actions take more than one clock interval.

Show time as milliseconds

Check this box to display and edit time in the graphical editor as milliseconds and not seconds. This is purely for your convenience; if your sequence sections are all less than a second you will probably find it more convenient to use milliseconds.

Sequence uses digital output bits

Check the boxes for the dedicated digital outputs that you will use. Only these outputs will appear in the editor, reducing visual clutter. If you do not require any digital outputs, clearing all the check boxes will save an instruction at the start of each sequencer section.

Sequence uses DAC outputs

Check the boxes for the Digital to Analogue Converters (voltage output devices) that you will use in your sequence. Unused DACs are not included in the graphical editor (to reduce visual clutter) and no sequence code is generated for them, which saves instructions at the start of each sequence section.

If you use one or more DACs for arbitrary waveform output only do not include them here unless you want to be certain that they have a defined value when you enter a sequence section. All 1401s have at least DACs 0 and 1; the Power1401 has DACs 0 to 3. Top boxes allowing up to 8 DACs can be added to Power1401s.

DAC full-scale, zero and units

You can define the DAC outputs in units of your choice. 1401 DACs normally have a range of ± 5 Volts, but ± 10 Volts systems exist. Set **Units** to the units you want to use. Set **Full-scale** to the value in these units that corresponds to the maximum DAC output. Set **Zero** to the value in your units that corresponds to a DAC output of 0 Volts. For a ± 5 Volt system calibrated in Volts, set **Full-scale** to 5, **Zero** to 0 and **Units** to v. If you want the output in millivolts, set **Full-scale** to 5000, **Zero** to 0 and **Units** to mV.

Absolute pulse levels

The DAC pulses take their starting level as the current DAC value at the pulse start time. The DAC then changes to another value, then back to the original level. Normally you define pulses in terms of the pulse amplitude relative to the starting level and all pulses add. If you check this box, then you set the absolute level for the DAC to change to. Data is written for each sequencer section that has any output defined or that has a key set.

Write as text sequence...

You can use this button to output the graphical sequence as a text sequence. You might do this if the graphical interface almost does what you need and you need to hand-edit a few extra instructions, if you want to run multiple graphical sequences, or if you need to save the sequence for documentation purposes. A file selector dialog opens for you to choose a name for the .PLS file.

Clear graphical editor...

This wipes all graphical sequences. You must confirm this action as you cannot undo it. This also clears the keys associated with each section.

Graphical editor...

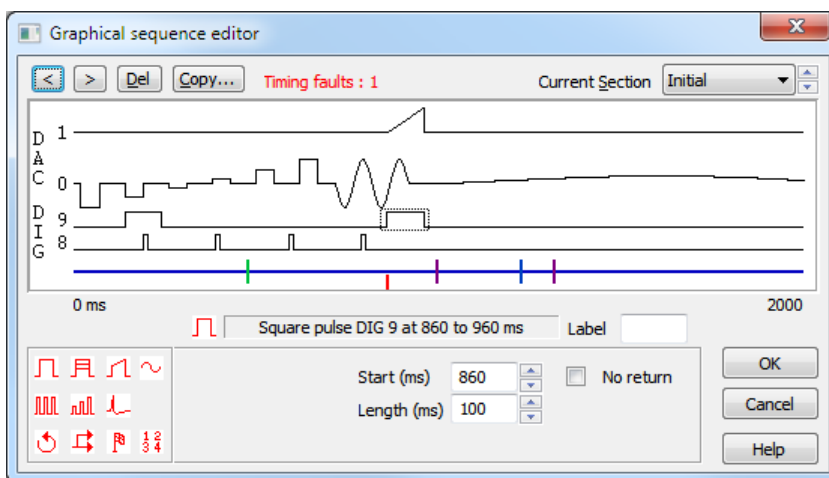
This opens the editor so you can make changes to the sequence.

Graphical editing

To open the graphical editor, select Use graphical editor and click the Graphical editor... button in the Sequencer tab of the Sampling configuration dialog. You can resize the dialog by clicking and dragging an edge. Double-click the title bar to maximise the dialog, double-click again to minimise it. The editor window has five areas:

1. a window with a graphical representation of the output sequence
2. above the graphical window are controls to iterate through and delete selected items, a message area and the Current Section selector
3. the lower left-hand corner holds a palette of items to drag into the graphical window
4. the lower right hand corner has control buttons
5. the settings for a selected graphical item lie between the palette and the buttons

The OK and Cancel buttons both close the dialog. OK accepts all changes, Cancel rejects all changes. The Help button displays the information you are reading!



Graphical editor

Sections

There are 27 sections: Initial and Section A through Section Z. The Current section field sets the section to

display and edit. The Initial section runs when the sequence starts; in some cases this may be the only section you need. The remaining sections are optional. A section displays a representation of the output for each DAC and digital output that is set for use in the **Sequencer** tab of the sampling configuration. There is a thicker line at the bottom for the control track, which holds all other sequencing actions.

Selecting items

Click an item in the graphical view to select it. A grey rectangle marks the selected item and the item settings appear below the display. The < and > buttons at the top left select the previous and next item on the current track; they are very useful when items overlap.

Section settings

Click on the control track clear of any dropped items. You can now set the associated key, the section length, number of repeats, action when the repeats are done, and a comment that is displayed in the sequencer control panel to identify sections with keys. You can set a section length up to 10000 seconds and up to 1000000 repeats! If you want a section to repeat forever, set the repeat count to 0. The Then field determines what happens when the section ends. You can stop the sequencer with **Halt**, or select another section to run.

Length (ms)	2000	Repeats	1	Then	Halt
Comment	Pulse outputs				

Key field

In an empty graphical sequence, no section has a key assigned. If you assign a key to a section, that section can be activated by a user keypress (if enabled) and the section will be listed in the sequencer control panel together with the section comment. You can set the key to any of A-Z, a-z or 0-9 (upper and lower case are different). Only one section is allowed to use any particular key.

Adding and deleting items

The graphical palette at the bottom left-hand corner of the dialog contains all the items that you can add to the display. Move the mouse over an item so see a short description. Click an item and drag it to a suitable track, then release to add it to the section. The **Del** button on the top line of the dialog removes the selected item. You cannot remove the control track or the lines that represent the initial state of the DACs and digital outputs.

Dragging and duplicating an item on a track

To move an item, click on it and drag to the destination and release. To duplicate an item, hold down the **Ctrl** key, click on the source item and drag it to the destination position; you must keep **Ctrl** held down when you release the mouse button or the duplicate operation will become a drag.

Timing faults

The sequencer attempts to match the timing you request. If this cannot be done, timing conflicts are marked by a red vertical line below the control track and the number of conflicts is given in the message area. You will get a conflict if you try to position any action at the start of a section. This is because the first instructions in a section set the initial digital output and DAC state. You can choose to ignore timing faults; the sequence will run with the changes as close to the requested time as possible. The sequencer time resolution field of the main sequencer page sets how close the next instruction can be to the place you asked for.

Copy section

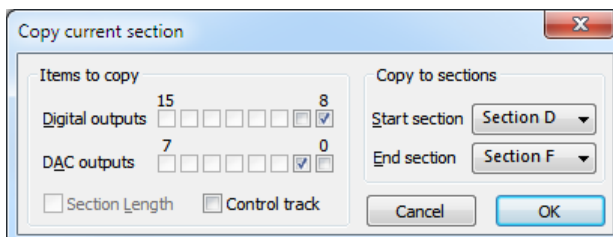
The **Copy** button opens a dialog where you can copy items in the current section to a range of sections.

Setting initial DAC and digital levels

To set the initial DAC and digital levels, click on a track clear of any added items. The initial digital output level is 0 or 1; the initial DAC level is in the units set by the **Sequencer** tab of the **Sampling configuration**. Every section starts with the instructions that set these levels, so output changes caused by them will be as close to time 0 in a section as possible. The fact that these levels may not be set at exactly time zero is not considered a timing fault.

Copy section dialog

The Copy button opens a dialog where you can copy items in the current section to a range of sections. This makes it easy to create lists of similar stimuli. The left-hand side of the dialog sets the items to copy; the right-hand side sets the destination. If you clear the Section Length check box, the length of the target sections does not change, otherwise the target sections are set to the length of the source section. If you check the Control track box, the Section Length box is ignored and the length is always copied. For any item selected for copying, the corresponding item in the target section is first deleted, then the source item is copied.



Graphical palette



The palette contains 11 items you can drag and drop in the graphical window to generate a sequence. There are three dropping zones: a digital output, a DAC output and the control track. Items will only drop onto suitable targets. For example, you cannot drop a sinusoid on a digital track. There is no need to drop the items at exactly the right time point; you can edit the position afterwards.

The palette is replaced by an error message if you set an illegal value for a pulse; you must fix the error before you can add more items.

Dragging and duplicating an item on a track

To move an item, click on it and drag to the destination and release. To duplicate an item, hold down the `Ctrl` key, click on the source item and drag it to the destination position; you must keep `Ctrl` held down when you release the mouse button or the duplicate operation will become a drag.

Common editable fields

When you select an item in the graphical window you can edit the fields that relate to it. The following are described here to avoid repeating the descriptions.

Label

This field is normally blank. Use it to label the selected item so that you can branch to it. A label can be up to 6 alphanumeric (A-Z, 0-9) characters long and is case insensitive; `abc23` and `ABC23` are the same. Labels must be unique in each section, but you can have the same label in a different section.

You cannot set a label for the initial levels of the digital and DAC tracks or for the control track as these items all start at time 0 and can be branched to by referring to the section name. You cannot set a label for arbitrary waveform output as this would prevent an important optimisation required when the sequence is generated.

Start time/At

All digital and DAC change items have a **Start time** field and all items on the control track have an **At** field. The time is in units of seconds or milliseconds, as set by the **Graphical editor settings**, and is relative to the start time of the sequence section. There is a spin control to nudge the time on or back by the sequencer time resolution.

If you add items to the control track that take an unknown time to complete, for example a random delay or a wait for an input signal to achieve a set value, the **At** time determines where the items are drawn in the graphical editor. In this case the sequencer will maintain the time intervals between items wherever possible.

For pulses, ramps, sinusoidal and arbitrary waveform output, the sequencer attempts to produce the first output change at exactly the time you specify. For other item types, the sequencer attempts to run the first instruction of the item at the specified time.

Length

Several items have a length in seconds or milliseconds. For all the pulse types, this is the period for which the

pulse changes to the amplitude or level you set before it reverts to the original level. For a ramp and a sinusoid, this is the length of the output. For arbitrary waveform output, this is set to the length of the arbitrary wave you associate with the command. You can set it to be less than the length of the wave, in which case the output is stopped after the time you have set. There is a spin control to nudge the time on or back by the sequencer time resolution.

Interval

For the pulse train items, this is the time between pulse starts in seconds or milliseconds.

Level (units)/Size (units)

These fields are used with DAC pulse outputs. The field you see depends on the state of the **Absolute pulse levels** check box in the **Graphical editor settings**. The **Size** field sets the amplitude of a pulse; the **Level** field sets the absolute level of a pulse. You set the value in the DAC units set in the **Graphical editor settings**.

No return

Normally the output from single pulses, ramps and sinusoidal output returns to the background level at the end of the item. If you check this box, the output change produced by this item is not removed at the end of the item. To indicate this, the grey rectangle surrounding the item extends to the end of the section. You could use this to ramp a DAC up to a value and leave it there.

Overlapping items

If digital pulses overlap, the result is the logical OR of the pulses.

If DAC items overlap on the same channel, the output depends on the state of the **Absolute pulse levels** check box in the **Graphical editor settings**. If it is clear, the result is the sum of the outputs. If it is checked, the output level is set by the last item in the overlap. There is an exception: arbitrary waveform output overrides all other items.

When adding pulses and pulse trains where the result would exceed the DAC range, the output is limited to the DAC range. However, pulses with an amplitude change on repeats can exceed the DAC range and wrap around. A value that goes off the top of the range will reappear at the bottom; a value that goes off the bottom of the DAC range will reappear at the top.

Graphical palette items

Some of the following descriptions say that a time value is 'limited to 31 bits of sampling clock ticks'. This is because the sequencer variables are 32-bit signed values. Values stored in them range from about minus 2 billion (-2,000,000,000) to plus 2 billion (2,000,000,000). At a time resolution of 1 microsecond, 2 billion ticks represents 2,000 seconds, which is usually sufficient. Other items may be limited to 31 bits of sequencer clock ticks for the same reason, but as the fastest sequencer rate is 10 microseconds per tick, this is unlikely to cause a problem.

Single pulse

You can drag this item onto either the digital or the DAC outputs. For a digital output, this sets the output to be the inverse of the initial level set for this output in this section. For a DAC output, you set the amplitude of the pulse with the **Size** field or the absolute level with the **Level** field.

Single pulse amplitude change on repeat

You can drag this item to a DAC output. It is for use in a repeated section. The **Size/Level** field sets the initial amplitude or absolute level of the pulse the first time the section runs. The **Change** field sets the amplitude change to apply on each subsequent repeat. You set the number of repeats by selecting the control track and editing the **Repeats** field. The changes are calculated in real time in the 1401. If the initial level plus the number of changes times the number of repeats exceeds the DAC range, the output wraps around.

Ramp

You can drag this item to a DAC output. The **From** and **To** fields set the initial and final amplitudes or levels of the ramp depending on the state of the **Absolute pulse levels** check box in the **Graphical editor settings**.

Sinusoid

You can request sinusoids on DACs 0 to 7 in the Power1401 mk II and -3. However, Micro1401s can only generate them on DACs 0 and 1 and the original Power1401 can use DACs 0 to 3.

The **Size (units)** field sets the sinusoid amplitude; this is not affected by the **Absolute pulse levels** check box. You can offset the sinusoid with the **Centre (units)** field. If the **Absolute pulse levels** check box is clear, the sinusoid and offset are added to the DAC value. If the check box is set, the DAC output is defined by the sinusoid and offset.

The **Period** field sets the time for one cycle of the sinusoid in seconds or in milliseconds. Alternatively, you can use the **Frequency (Hz)** field to set the frequency. The **Phase** field sets the initial phase in degrees. The output is a cosine, so a phase of 0 means start at maximum amplitude. A phase of -90 or 270 produces a sine output.

Pulse train

You can drag this to a DAC or digital output. It generates a train of pulses defined by the number of pulses (**Pulses** field), the length of each pulse (**Length** field) and the interval between each pulse and the next (**Interval** field) or the pulse **Frequency (Hz)** field. For a DAC output you can also set the amplitude with the **Size/Level** field.

This method is inefficient if you need to generate a large number of pulses as each pulse takes several instructions. You should consider setting up a single pulse as a section and repeating the section to create the pulse train.

Pulse train with varying amplitude

You can drag this to a DAC output. It generates a train of pulses defined by the number of pulses (**Pulses** field), the length of each pulse (**Length** field) and the interval between pulse starts (**Interval** field) or the pulse **Frequency (Hz)** field. You set the amplitude of the first pulse with the **Size/Level** field. The pulse size changes by the value in the **Change** field for each repeat.

This method is inefficient if you need to generate a large number of pulses as each pulse takes several instructions. You should consider setting up a single pulse with changing amplitude as a section and repeating the section to create the pulse train.

Arbitrary waveforms

Drag this item to the control track to start playing a waveform through the 1401 DAC outputs. You can play arbitrary waveforms through DACs 0-3 (or any combination of these DACs). The DACs used are set when you create the waveform and can be edited in the **Play Waveform** tab of the **Sampling** configuration. Create the arbitrary outputs with the **Sample** menu **Output Waveform** command or from a script.

If more than one waveform is defined and enabled in the sampling configuration, you are prompted to choose one from a list. Double-click the dropped item in the control track to redisplay the list. Each waveform is identified by a code; this is either a single alphanumeric character or two hexadecimal digits. When you set this in the **Key** field or by selecting a waveform, the **Length** field is set to the waveform length or to a lesser value if the wave is longer than the section or to 0 if there is no matching wave. Set the length shorter than the entire wave to truncate the output.

Arbitrary output takes appreciable time to set it up; times in excess of 10 milliseconds are possible. When generating the output, this set up is moved as far forward in the sequence section as possible so that the output starts at the exact time you set. If the preparation has not completed by the time you request output to start, the sequence stalls until it is ready.

When the arbitrary output ends, the DAC outputs are returned to the background level as soon as possible. If the arbitrary waveform has more than one channel, there will be one sequencer clock period between each DAC changing to the background level.

Wait for time, condition or variable

You can drag this item to the control track only. It pauses the sequence until a condition is met. The duration of this item may be unknown; it is drawn as if it were of zero width. You can select the following delay types:

Fixed delay

Set the time to wait in the **Delay** field. This is useful for short active periods separated by long delays.

Random delay

This holds the sequence for a time between **Min time** and **Max time** seconds or milliseconds. All delays between the minimum and maximum have equal probability (within the capabilities of the sequencer). You will get the best result for delay ranges that are long compared to the time resolution.

Poisson delay

This generates a delay with a Poisson statistic; the delay has the same probability of ending at any time during it. The **Time const** field sets the average delay length.

Digital input high/low

You can wait for a nominated digital input bit in the range 0-7 to be high or low. These are not the same bits as used for the event inputs. If you want to synchronise to a bit changing you must wait for it to be in the opposite state to the one you want first. If you need the sequence to perform actions while you wait, use a branch item. You can wait for combinations of input bits with the text sequencer.

Channel above/below/outside/within

You can wait for a nominated waveform channel sampled by the 1401 to be above or below a level set by the **Threshold** field, or to be outside or within a pair of levels set by the **Lower** and **Upper** levels fields. The levels are set in the channel units, as stored in the sampling configuration. If you subsequently change the channel type, the results will not be harmful, but the sequence will not operate as intended!

Next event

You can wait for the number of events set by the **Count** field to occur on a nominated event, marker or WaveMark channel that is sampled by the 1401. When this delay item is reached, the sequence notes how many events have been sampled on the channel and waits until the number increases by the count.

Time reached

Waits until the sample time reaches the set time. There is no wait if the time has already passed the set value.

Cosine phase 0

This item waits for the next time that cosine output on the nominated DAC channel passes through phase zero.

Variable comparisons

You can wait for various conditions based on sequence variables 1 to 25. The remaining variables are used to implement the sequence sections. Variable values are 32-bit integers. You can manipulate the variables in the **Variable arithmetic** item. Variables are described in detail for the text sequencer.

Event burst

This item monitors an event, marker or WaveMark channel sampled by the 1401 for a group of events with a user-defined maximum separation. The **Intervals** field sets the number of gaps to check and the **Max time** field sets the maximum acceptable interval. If any interval is greater than the maximum, the sequence starts the search again. For this to work well, the maximum interval must be significantly greater than the time resolution of the sequence. The total burst duration is limited to 31 bits of sampling clock ticks.

Branch on condition, probability or variable

With this item you can break the normal flow of the sequence and branch to a different section or to a label you have defined for an item in the current section. All branches have a **Branch destination** field in which you can select a section to branch to, or you can type the name of a label in the current section.

When you branch, the timing to the target may not be exactly what you expect. The sequence will take one or more steps to implement the branch and the target instruction may require preparatory steps. Such effects are small unless you use arbitrary waveform output where the preparatory steps can take several milliseconds. If you need the tightest possible control over branch timing you should consider using the text sequence editor. The branches you can set are:

Probability

Percent sets the probability of the branch, 0% never branches, 100% always branches.

Digital input high/low

The Bit to test field sets the digital input bit number in the range 0-7 to test.

Channel above/below/outside/within

You can branch if a nominated waveform channel sampled by the 1401 is above or below a level set by the Threshold field, or is outside or within a pair of levels set by the Lower and Upper levels fields. The levels are set in the channel units, as stored in the sampling configuration. If you subsequently change the channel type, the result is not harmful, but the sequence will not operate as intended! It takes two (or three for the outside/within cases) sequencer instructions to do the check, so make sure that the sequencer is running fast enough to detect the changes you seek.

Variable comparisons

You can branch on the result of comparing sequence variables with constant values and other variables. Some variables have special uses. Variable values are 32-bit integers. You can manipulate the variables in the Variable arithmetic item. Variables are described in detail for the text sequencer.

Unconditional

This always branches to the destination.

Time comparisons

You can compare a variable plus a time offset with the current time and branch on the result. You can set the variable to the current time (plus a time offset) with the variable arithmetic *current time* instruction.

Response with timeout

You can wait for a new data item in an event, marker or WaveMark channel sampled by the 1401. The branch is taken if a new item is detected within the timeout period. The timeout period is limited to 31 bits of sampling clock ticks.

Generate digital marker channel event

This adds an event to the Marker channel (if it is enabled). You can set the marker code with the Marker code field or check the Record data box to record the state of digital input bits 0 to 7 (these are not the same bits used for event inputs). The Marker code field should be set to one character or to two hexadecimal digits.

Variable arithmetic

Although the use of variables is more commonly done with the text editor, you can perform basic variable manipulation here. You can use variables 1 to 25. Variable values are 32-bit integers. In all cases, the variable that is changed is set by the Target var field. Where operations involve time (in sampling clock ticks) you must remember that sequencer variables are 32 bits and times can now be up to 64 bits. This means that once the sample time exceeds around 2 billion clock ticks, the values stored will no longer represent the time in a useful way (though you can still use differences of times as long as the difference is less than 2 billion sampling clock ticks). Operations are:

Set to value/variable

Replace the target variable with the contents of the Value field or of a variable.

Add/subtract value/variable

Adds or subtracts the contents of the variable or value.

Multiply by value/variable

Multiplies the target variable by the variable or value.

Random value

This replaces the target variable by a random number that is from 1 to 30 bits long, set by the Bits field. The possible values for an n bit number are 0 up to $2^n - 1$. For example, if the Bits field is 4, the possible results are 0 to 15.

Current time

The variable is set to the current sample time plus a time offset, in Spike2 clock ticks, as set in the Resolution tab of the Sampling configuration. As the current time can occupy more than 31 bits, the value stored in the

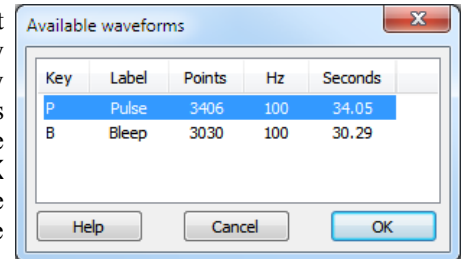
variable will, in general, only be useful for the first 2 billion sampling clock ticks. This is retained for compatibility with Spike2 version 7, but should be avoided in version 8.

Fixed time

The variable is set to a time, in Spike2 clock ticks, as set in the Resolution tab of the Sampling configuration. As variables can only store positive values up to 31 bits in length, this is only useful for a fixed time that is less than around 2 billion sampling clock ticks. This is here for compatibility with version 7, but should be avoided.

Select arbitrary waveform

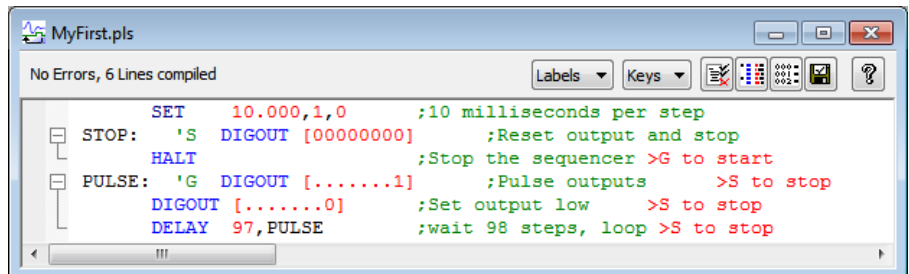
This dialog opens when you drag the arbitrary waveform output icon onto an output track and there is more than one arbitrary waveform to choose from or when you double click an arbitrary waveform output item in an output track. You can add items to this list from the Offline waveform output dialog accessed from the Sample menu. Select one of the waveforms in the list and click OK to add the wave into the sequencer output or click Cancel to close the dialog. You can adjust the length of the played data in the graphical editor.



The text editor

The output sequencer text editor gives you full control over the output sequencer. The price you pay for this is that you become responsible for calculating the timing of everything. Unlike the graphical editor, where you can deal with items like pulse trains, in the text editor you must construct pulse trains edge by edge; fortunately, this is not as difficult as it sounds. Even if you plan to use the graphical editor, some understanding of how you would program with the text editor is useful as it will help you understand why some features of the graphical editor work in the way they do (as all graphical sequences are converted to text sequences before they are used).

The text output sequence is stored as a text file with the extension .pls. You can open existing Spike2 output sequence files or create new ones with File menu New... If the folding margin is displayed as in this example (if not, see the



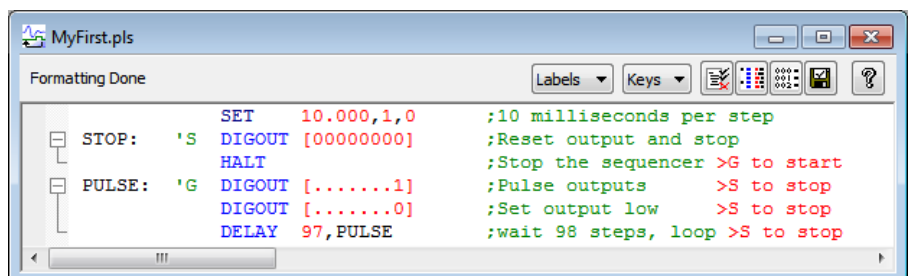
View menu Folding command and select a folding style), you will see that you can fold up the window text based on the keyboard codes as folding points. The Keys and Labels drop downs show a list of all the keyboard codes and labels in the file; select one to navigate to them. If you load a sequence from a read only medium or the file is marked read only, you are not allowed to change it in the editor.

There are five buttons at the top of the window:



Format

This aligns the labels, key codes, instructions and any arguments, output text and comments and removes step numbers. It will also do basic syntax checking of the sequence; undefined labels are not flagged but there must be no other errors. If there

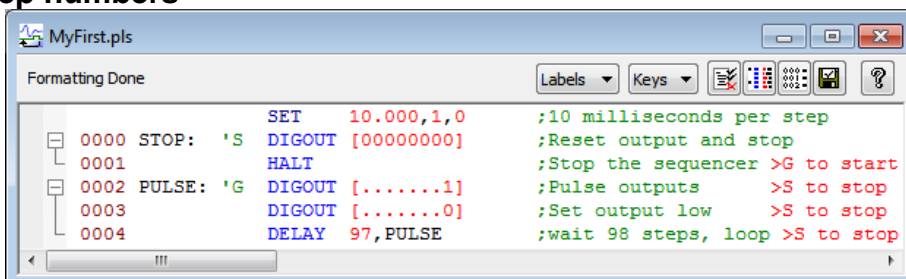


are errors, the first offending line is flagged and formatting stops. You will normally want to have your sequences formatted as it makes them much easier to read and understand. However, it is usually much quicker to type sequences in unformatted, then use the Format command to tidy them up.



Format with step numbers

This does the same job as the Format button, and also starts each line with the step number. Step numbers can be useful as they give an indication when you are running out of space (there are a maximum of 8191

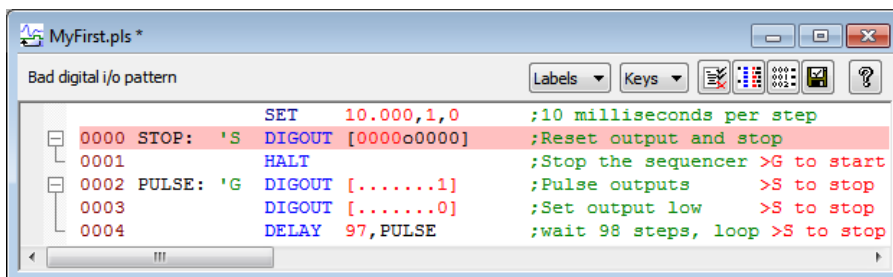


instructions allowed) and can pinpoint the line where your sequence is not behaving as you expect. When your sequence is running, the sequencer control panel displays the current sequencer step. A number at the start of a sequencer line is completely ignored when sequences are compiled, so you need not worry about editing a sequence that has step numbers. Just format the sequence to clean up the changes.



Compile

This checks the sequence to make sure that it is correct with no labels missing or duplicated and no duplicated key codes. The picture shows a sequence with a simple error (the o in the second line should be a zero). The line in error is



marked by changing the background colour and an explanatory message is shown at the top of the screen. Any edit to the sequence will remove the background colour change. The compiler stops at the first error it finds. If the compiler completes without error is display the number of sequencer lines (instructions) that were compiled. The compiler always adds a HALT instruction to the end of the sequence, so the count will be one more than the number of listed lines.



Current

This compiles the sequence and if it compiles without error, the sequence is saved. If there is any error, it is reported and the command ends. If there is no error, what happens next depends on whether Spike is sampling data or not:

Not sampling The name of the sequence file is set in the sampling configuration and the sequence will be used when you next sample data. You can also set the output sequence file from the Sampling Configuration dialog accessed from the Sample menu.

Sampling Spike attempts to replace the current sequence. This will fail if there is no current sequence, or if the sequence or table size is larger than the current size or the size reserved in the Sequencer tab of the Sampling configuration dialog. If a sequence is replaced, it runs from the first instruction. Any variables that are declared with initial values are updated as are all initialised table values. The Sampling configuration is not updated with the name of this sequence file.

You can also change the current sequence file from the Sample menu.

A message indicating the outcome of the operation is displayed at the top of the window.

Help

This is the Help button. It opens a window holding a list of the sequencer topics for which help is available. You can copy and paste text from the help window into your sequence.

Loading sequencer files for sampling

The name of the output sequencer file to use during sampling is part of the sampling configuration. The file name, including the path to the folder containing it, must be less than 200 characters long. You set the file name either with the **Current** button, as described above, or in the **Sampling Configuration** dialog. When you start sampling, Spike2 searches for any output sequence file named in the sampling configuration.

Spike2 compiles output sequence files whenever you use them. If a file contains errors you are warned and the file is ignored.

Getting started

The sequencer runs instructions in order unless told to branch. It can be re-routed during data acquisition by associating an instruction with a key on the keyboard. Each time the key is pressed or the associated sequencer control panel button is clicked or the script language `SampleKey()` command is used, the sequencer jumps to the associated instruction. Spike2 records keys pressed during sampling, so you have a record of where in the document you switched to a new portion of the sequence.

Here is a simple sequence that will pulse digital output bit 0 for 10 milliseconds once per second. You can start and stop the pulses with keys or by clicking buttons. You will find this in `Spike6\Sequence\MyFirst.pls` to save you typing it in.

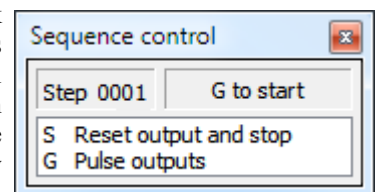
```

      SET      10          ;10 milliseconds per step
      'S DIGOUT [00000000] ;Reset output and Stop
      HALT          ;Stop the sequencer >G to start
PULSE: 'G DIGOUT [...1]  ;Pulse outputs >S to stop
      DIGOUT [...0]      ;Set output low >S to stop
      DELAY 97,PULSE     ;wait 98 steps, loop >S to stop

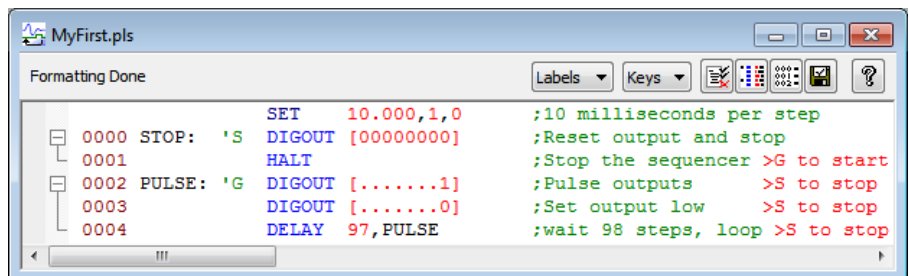
```

Open `MyFirst.pls` and click the **Check and make current sequence** button at the upper right of the window (leave the mouse over each button for a second or so to see the descriptive text). Open the **Sampling configuration** dialog, and set a configuration with one event channel on port 0. To record the output pulses you must connect the digital output to the event input. This is easy with the Micro1401 and Power1401 as you can connect Digital output 0 to Event input 0 on the front panel. You do not need to make the connection to follow this description.

Click the **Run now** button in the sampling configuration, and then click the **Start** button. The sequencer control panel is now visible. It displays **Step 0001** in the top left window and **G to start** to the right. If the control panel is docked there are two buttons labelled **S** and **G**, otherwise you can see the text **S Reset output and stop** and **G Pulse outputs**. To start the output, click on the **G** or type **G** on the keyboard. While it runs, the display will change to **Step 0004** (with occasional flickers) and **S to stop**. If you have connected the digital output to the input, you will see your pulses recorded, once per second. Click the **S** or type **s** on the keyboard to stop the output.



As this is our first sequence we will explain it in detail. Click the **Format and add step numbers** button at the top of the sequencer window. All the lines get a step number except the first. This is the step number displayed in the control panel. You can remove step numbers with the **Format** button. Lines that get a number are part of the



sequencer window. Lines that get a number are part of the

sequence run by the 1401. The `SET` line tells the sequencer how fast to run, in this case 10 milliseconds per step, and is not part of the sequence run by the 1401. A `SET` or `SCLK` directive is usually the first line in the sequence.

The next line (step 0000) is the first to run when you start sampling. The `'S` means *jump to this line every time the S key is pressed during sampling* (as long as the data window is the current window). To allow you a wide choice of keyboard jumps, lower case `s` and upper case `S` are treated as different keys.

The instruction on this line is `DIGOUT [00000000]`, which sets 8 digital outputs (bits 15 to 8 of the digital output) to the low state, nominally 0 Volts. The remainder of the line is a comment. Because there is a keyboard jump set for this line, the comment is also used as a label for the sequencer control panel.

The next line, (step 0001) holds a `HALT` instruction. This stops the sequencer and nothing will happen until the sequencer is told to jump to one of the steps labelled with a key code. The sequencer control panel displays the text to the right of the `>` for the current step. When the sequencer halts, it stays at step 0001, so you see the message `G to start`.

Steps 0002 and 0003 set digital output bit 8 high (nominally 5 Volts) and low again. The sequencer is running at 10 milliseconds per step, so the pulse is 10 milliseconds wide. Step 0002 starts with a label, `PULSE:`, so we can branch back here in the next step.

Step 0004 has the instruction `DELAY 97,PULSE` to make the sequence wait for 97 extra step times in addition to the step time that every instruction takes, and then branch to the instruction labelled `PULSE`. This instruction takes 980 milliseconds, and together with the 20 milliseconds taken by steps 0002 and 0003, this makes 1000 milliseconds for the loop. Instead of 97 we could have written `s(1)-3`. The `s(1)` function returns the number of sequencer steps in 1 second.

Time delays can also be implemented with the `TICKS` instruction, which gives you access to the underlying tick count used by the Spike2 sampling.

The remainder of this chapter is organised as a reference manual for the sequencer instructions. You can find more information about the output sequencer in the *Spike2 Training Course* manual.

Sequencer compiler error messages

When you use the `Format` or `Compile` buttons in the sequence editor, Spike2 displays the result of the compilation or format operation in the message bar at the top of the window. The messages report either successful operation or the cause of the problem.

```
No errors, N lines compiled
```

Your sequence has been checked and is syntactically correct. This means that it will certainly run, but it is up to you to ensure that the sequence of codes produces the desired effect. If your sequence runs off the end of the defined instructions, it will stop as the compiler adds a `HALT` instruction to the end of every sequence. The number of lines compiled is a count of the number of instructions generated, which is one more than the number in the sequence text as Spike2 adds a `HALT` instruction to the end of the sequence.

Formatting done

The format operation has completed and all lines of instructions are syntactically correct in themselves. There is no check that a label referenced in an instruction exists.

Invalid label format, or same as Opcode, VAR, CONST or function

A colon was found, indicating a label, but the characters before the colon are not legal. A label starts with an alphabetic character (`A-Z`) and is followed by alphanumeric characters (`A-Z` and `0-9`) and is terminated by a colon. Case is not significant in labels. The label can be up to 8 characters long. It must not be the same as an instruction name, a variable name, a constant name or a built-in function.

Invalid key definition

A quote mark was found, but there was no acceptable character following the quote, or there was more than one character.

Unknown command used

A group of characters was found in the correct position to be an instruction, but they were not recognised. Change them to a correct instruction mnemonic. This can also be caused by a missing colon at the end of a label.

The first/second/third/fourth argument is invalid

The instruction argument is either missing, invalid or was too long to be correct. If the argument included a table reference, the offset may be too large or too small.

Internal error, contact CED

This is caused by a serious system error. If you can reproduce this problem please contact CED for advice.

Out of memory

Spike2 has run out of memory while processing the sequence. Close any non-essential windows in the Spike2 application and try again. If this does not allow compilation, close any other applications and retry. As you are most unlikely to exhaust your memory when compiling it may mean that something has gone badly wrong.

Label or table size defined twice

A label has been defined on more than one line. Remember that the labels are converted to upper case, so two labels that differ in case only will cause this error. This error is also given if you use TABSZ more than once in a sequence.

Unexpected characters at end of line

The most common cause of this error is a missing semicolon to introduce a comment.

Couldn't find branch destination

The label used by the instruction is not defined anywhere in the sequence. Define the label, or correct the label name.

Invalid numeric value

A numeric argument was expected, but an invalid number or one that was outside the permitted range for the argument was found.

Probability out of range

The probability field of the BRAND instruction must be set to a number in the range 0 to less than 1.0000000000; correct the number.

Bad digital i/o pattern

Whenever a digital i/o pattern is expected, exactly 8 characters must be present, one for each bit of the digital i/o. The characters are restricted to "0", "1" and ".", plus "c" for DIGIN and "i" for DIGOUT.

A label, key or display string but no instruction

Labels, key definitions and display strings may only occur on lines that have an instruction. If you really want an instruction that does nothing, use NOP. Blank lines and lines consisting entirely of comments are allowed and ignored.

Too many instructions

You have defined more than the maximum allowed number of instructions (8191), so you must shorten your sequence. You may be able to significantly shorten your sequence by using variables and the SampleSeqVar() script instruction. You can also split a sequence into sections and load these one at a time.

Input line too long; shorten and try again

Each line of the file must be no more than 100 characters long. Shorten the offending line and try again. This limit in previous versions of Spike2 was less, so it is a good idea to keep to short lines wherever you can.

Unexpected end of file encountered

This shouldn't happen and probably means that something has gone badly wrong.

Variable name too long, unknown, badly formed or missing

A variable name used to replace the v1 to v256 built-in names is incorrectly formed.

Please use a single character or 2 hexadecimal digits as wave code

The waveform codes for the WAVEGO command must either be a single character or two hexadecimal digits. A hexadecimal digit is one of the characters 0-9 or a-f.

Bad branch code: C=Cycles, A=Area, S=Stopped, T=Triggered, W=Wait for WAVEGO

The branch code for the WAVEBR command was not one of the allowed characters listed, or the branch code was more than one character.

Bad start or stop code: S=Stop now, C=one more cycle, T=Trigger now

The code given for the WAVEST command was not one of those listed, or there was more than one character in the code.

Bad flags: T=Triggered, W=Wait until hardware is ready

The flags for the WAVEGO command were not those listed, or more than two flags were used.

Variable name already defined as Opcode, label, function or CONST

Previous versions of Spike2 allowed a variable name to be the same as a label or an Opcode. This is no longer allowed. Also, the name is not allowed to be the same as a built-in function, for example vAngle or the same as a CONST item. Change the variable name.

Label already exists as Opcode, variable, function or CONST

Previous versions of Spike2 allowed a label to be the same as a variable name or an instruction name. This is no longer allowed. The name may not match a built-in function or a constant. Change the label.

The DAC value exceeds the DAC output range

You have entered an expression for a DAC value that when scaled and offset by the values in the SET directive, produces a result that exceeds the DAC range. Check that the SET directive scale and offset are correct and that you have not mistyped the value.

More table entries than set by TABSZ

You have used the TABDAT directive to create table entries. There is either no TABSZ directive, or you have generated more table data than you allocated with TABSZ.

Compatibility with previous versions

Sequences from all previous versions still work, with the following provisos on sequences written for Spike2 version 3 and earlier:

1. Variables and labels may not share names nor be the same as instruction codes.
2. DELAY 0 no longer hangs up for a long time. It now has the same effect as NOP.
3. If a version 3 sequence relied on looping back to the start after 256 steps, this no longer works as all sequences have a HALT instruction automatically added to the end.
4. BRAND set the probability to an accuracy of one part in 256. It now sets the probability to a very high accuracy. This might affect your results.
5. DACn and ADDACn now insist that any value you use lies within the range of the DAC.
6. ADDACn previously expressed your increment as a 16-bit integer. It now generates a much more accurate 32-bit integer. However, this means that ramps generated with ADDACn may have a different slope (and probably much closer to the intended slope).

If you want to write a sequence that will run on previous versions of Spike2, either refer to the manual for the previous version, or write the sequence using the previous version to be certain of complete compatibility. Use of any of the following features will prevent the sequence working on version 6 or earlier:

1. Use of more than 1023 sequencer instructions or more than 64 variables.
2. Use of the ASz() or VSz() expressions.
3. Use of a label in an expression
4. Use of the JUMP (Vn) or JUMP LB(Vn) instructions
5. Use of new instructions: TABADD, TABSUB, ABS, AND, ANDI, OR, ORI, XOR, XORI
6. Use of #include to include other files.
7. Use of the = directive to define a constant.
8. Use of more than DACs 0-3 in the sinusoidal output or RAMP instructions.

If you use the sTk64h() or sTk64l() instructions or refers to vTick0L or vTick0H, your sequence will only run in version 8.

Instructions

These are the output sequencer instructions in Spike2 versions 5, 6 and 7. Instructions in brackets are obsolete and should be avoided in new sequences.

Digital (TTL compatible) input and output

DIGOUT	Write to digital output bits 15-8
DIGLOW	Write to digital output bits 7-0
DIBNE, DIBEQ	Read digital input bits 7-0, copy to V56 test and branch
DISBNE, DISBEQ	Test last read digital inputs (in V56) and branch
WAIT	Wait for a pattern on the lower 8 digital inputs
(DIGIN)	Read and test digital inputs
(BZERO, BNZERO)	Branch as result of DIGIN test

DAC (waveform/voltage) outputs

DAC	Set DAC value (for DACs 0-7)
(DACn)	Version 3 compatible, set DAC n (0-3) to a value
ADDAC	Increment DAC by a value
ADDACn	Version 3 compatible, increment DAC n (0-3) by a value
RAMP	Set automatic DAC ramping to a target value

Sinusoidal waveform output

SZ	Set the cosine output amplitude (CSZ, DSZ)
SZINC	Change the cosine output amplitude (CSZINC, DSZINC)
RATE	Set the cosine angular increment per step (CRATE, DRATE)
RATEW	As RATE, but waits for phase 0 (CRATEW, DRATEW)
ANGLE	Set cosine angle for the next step (CANGLE, DANGLE)
WAITC	Branch until cosine phase 0 (CWAIT, DWAIT)
RINC	Change the cosine angle increment per step (CRINC, DRINC)
RINCW	As RINC but waits for phase 0 (CRINCW, DRINCW)
PHASE	Defines what phase 0 means (CPHASE, DPHASE)
OFFSET	Offset for sinusoidal output (COFF, DOFF)
CLRC	Clear the new cycle flag (CWCLR, DWCLR)

General control

DELAY	Do nothing for a set number of steps
DBNZ	Decrement a variable and branch if not zero
(LDCNTn, DBNZn)	Load counter 1 to 4 (V61-V64), decrement, branch if not zero
Bxx	Compare variables and branch (xx = GT, GE, EQ, LE, LT, NE)
CALL	Branch to a label, save return position
CALLV, (CALLn)	Like CALL, but load a variable (counter 1-4) with a value
RETURN	Branch to instruction after last CALL, CALLV or CALLn
JUMP	Unconditional branch to a label
HALT	Stops the sequencer and waits to be re-routed
NOP	This does nothing for one step (NO OPERATION)

Variable arithmetic

ABS	Take the absolute value of a variable
ADD	Add one variable to another
ADDI, (ADDIL)	Add a constant value to a variable
MOV	Copy one variable to another
MOVI, (MOVIL)	Move a constant value into a variable
MUL, MULI	Multiply two variables, multiply by a constant
NEG	Move minus the value of a variable to another
SUB	Subtract one variable from another
DIV, RECIP	Division and reciprocal of variables

Variable logic

AND, ANDI	Bitwise AND of variables, variable and constant
OR, ORI	Bitwise OR of variables, variable and constant
XOR, XORI	Bitwise exclusive OR of variables, variable and constant

Table support

TABLD, TABST	Load a register from the table and store a register to the table
TADADD, TABSUB	Add a table value to a variable or subtract it from a variable
TABINC	Increment a register and branch while within the table

Access to data capture

REPORT, MARK	Simulate an external E1 pulse to record/set a digital marker
CHAN	Get the latest waveform value or event count from a channel
TICKS	Load a variable with the time in Spike2 time units

Randomisation

BRAND, (BRANDV)	Random branch with a probability
MOVRND	Load a variable with a random number
(LD1RAN)	Load counter 1 (v61) with a random number 1-256

Arbitrary waveform output

WAVEGO	Start or prepare arbitrary waveform output area
WAVEBR	Test arbitrary waveform output and branch on the result
WAVEST	Start or stop arbitrary waveform output

Instruction format

Blank text lines and lines with a semicolon as the first non-blank character, are ignored. Instructions are not case sensitive. Each instruction has the format:

```
num lab: 'keycode arg1,arg2,... ;Comment >display
```

num	An optional step number in the range 0 to 8190, for information only. A number at the start of an instruction line is ignored.
lab:	An optional label, up to 8 characters long followed by a colon. The first character must be alphabetic (A-Z). Labels are not case sensitive. Labels may not be the same as instruction codes, variable names, constants or expression function names. Labels can be used in expressions, and have the value of the instruction number in the sequence; the first instruction is number 0.
'key	In this optional field, <i>key</i> is one alphanumeric (a-z, A-Z, 0-9) character. When this character is recorded as a keyboard marker during data capture, the sequencer jumps to this instruction. Each <i>key</i> can occur once. Upper and lower case are distinct. The <i>key</i> appears in the sequencer control panel.
code	This field defines the instruction to be executed. It is not case sensitive.
arg1,...	Instructions need up to 4 arguments and are separated by commas or spaces. These are described with the instructions. If an argument can be represented in different ways, they are separated by vertical bars (read as "or"), for example: <i>expr</i> <i>Vn</i> [<i>Vn+off</i>]. In this case, the argument can be an expression, a variable or a table reference.
comment	The text after the semicolon is to remind you of the reason for the instruction. If a <i>key</i> is set, this comment also appears in the sequencer control panel.
>display	When a sequence runs, text following a > in a comment is displayed in the sequencer control panel to indicate the current instruction. Spike2 is notified every few milliseconds of the current instruction and the control panel is updated whenever there is free time, so the display is only a sampling of what is going on. The special combination >" means use the same display text as for the previous instruction. This can save you some typing when a group of instructions in a loop need the same display. If there are several instructions in a row that use this, Spike2 searches backwards until it finds an instruction with a display string that is not >". The special combination >= means make no change to the display string. This might be used in a code section that is called as a subroutine from several different places.

Expressions

Many instructions allow the use of an expression in place of a constant value, indicated by `expr`. An expression is formed from constants, numbers, labels, round brackets (and), the operators +, -, * and /, and sequencer expression functions. Labels have the value of the instruction number in the sequence; the first instruction is numbered 0.

The operators * and / (multiply and divide) have higher priority than + and - (add and subtract). This means that $1+2*3$ is interpreted as $1+(2*3)$ and not as $(1+2)*3$. Apart from this, evaluation is from left to right unless modified by brackets.

The sequence compiler evaluates expressions as real numbers, so $3/2$ has the value 1.5. If `expr` is used as an integer, for example `DELAY expr`, it is rounded to the nearest integer. Values in the range 3.5 to 4.49999... are treated as 4. Before version 4.06 the result was truncated, so 3.0 to 3.9999... was treated as 3.

Sequencer expression functions

These functions can be used as part of expressions to give you access to Spike2 clock and sequencer step timing and to convert between user units and DAC and ADC values and to calculate table indices.

<code>s(expr)</code> <code>ms(expr)</code> <code>us(expr)</code>	The number of sequencer steps in <code>expr</code> seconds, milliseconds or microseconds. For example, with a step size of 200 milliseconds, <code>s(1.1)</code> returns 5.5. This is often used with the <code>DELAY</code> instruction. Each instruction uses 1 step, so use <code>DELAY s(1)-1</code> for a delay of 1 second.
<code>sTick(expr)</code> <code>msTick(expr)</code> <code>usTick(expr)</code>	The number of Spike2 sample clock ticks in <code>expr</code> seconds, milliseconds and microseconds. The result is in the same units as returned by the <code>TICKS</code> instruction, so <code>TICKS v1,sTick(1)</code> sets <code>v1</code> to the value that <code>TICKS</code> will return 1 second later. The result is a signed 32-bit value, so is limited in magnitude to 2 billion (2000000000) sample clock ticks. If you run sampling at a resolution of 1 microsecond, this is a range of -31 minutes to 31 minutes. At 2 microsecond resolution it is plus or minus an hour, and so on.
<code>sTk64h(expr)</code> <code>sTk64l(expr)</code>	New at Spike2 version 8.00. These converts a time in seconds into Spike2 sample clock ticks as a 64-bit number, then returns either the upper 32 bits (<code>sTk64h</code>) or the lower 32 bits (<code>sTk64l</code>). If you need to do 64-bit arithmetic with these values you should be aware that the lower 32 bits is effectively an unsigned number, so if you add a 32-bit value to it you may need to detect if the result overflowed, and if it did overflow add 1 to the upper 32 bits. There are currently no 64-bit arithmetic instructions in the sequencer instruction set, but you can emulate them (with some effort) by looking at the signs of the values before and after addition.
<code>Hz(expr)</code>	The angle change in degrees per step for a cosine output of <code>expr</code> Hz. For a 2 Hz cosine on DAC <code>n</code> , use <code>RATE n,HZ(2)</code> . To slow the current rate down by 0.1 degrees per step use <code>RINC n,HZ(-0.1)</code> . Use in <code>RATE</code> , <code>RATEW</code> , <code>RINC</code> and <code>RINCW</code> instructions.
<code>VHz(expr)</code>	The same as <code>Hz()</code> , but the result is scaled into the 32-bit integer units used when a variable sets the rate. <code>MOVI v1,VHZ(2)</code> followed by <code>RATE n,v1</code> will set a 2 Hz rate.
<code>VAngle(expr)</code>	Converts an angle in degrees into the internal angle format. The 32-bit integer range is 360 degrees. The result is <code>expr * 11930464.71</code> . For use with <code>ANGLE</code> and <code>PHASE</code> .
<code>VDAC16(expr)</code>	Converts <code>expr</code> user DAC units so that the full DAC range spans the full range of a 16-bit integer. Use with variables for the obsolete <code>DACn</code> and <code>ADDACn</code> .
<code>VDAC32(expr)</code>	Converts <code>expr</code> user DAC units into a 32-bit integer value such that the full DAC range spans the 32-bit integer range. Use this to load variables for the <code>DAC</code> and <code>ADDAC</code> instructions.
<code>ASz(expr)</code>	Converts <code>expr</code> user DAC units into a value suitable for use with the <code>SZ</code> and <code>SZINC</code> commands. That is, it converts an amplitude or amplitude change in user DAC units to a value in the range 0 to 1. It does this by dividing the value you supply by the (DAC scale factor * 5).
<code>VSz(expr)</code>	Converts <code>expr</code> user DAC units into a variable value suitable for use with the <code>SZ</code> and <code>SZINC</code> commands. That is, it converts an amplitude or amplitude change in user DAC units to a

value in the range 0 to 32768. It does this by multiplying the value you supply by 32768/(DAC scale factor * 5).

- TabPos() The number of table data items defined at this point in the sequence by TABDAT. This is also the index of the next table data item to use. You can use this to define constants that reference table indices.
- DRange() The DAC output range (usually 5.0 for a ±5 Volt system or 10.0 for a ±10 Volt system). When the sequence is built for use during sampling, this value is taken from the DAC output range of the attached 1401, otherwise, this is taken from the Voltage range set in the Edit menu Preferences option.

Used with care, the built-in functions allow you to write sequences that operate in the same way regardless of the sequencer step time or DAC scaling values.

Variables

You can use the 256 variables, v1 to v256, in place of fixed values in many instructions. In the sequencer command descriptions, vn indicates the use of a variable. Where a variable is an alternative to a fixed value expression we use `expr|vn`. Variables hold 32-bit signed integer numbers that you can set and read with the `SampleSeqVar()` script command. There are instructions that can be used to perform arithmetic on variables.

Variables with pre-defined uses

Some variables have specific uses and pre-defined symbolic names (which are easier to remember than v1 to v256). If you use these variables for other purposes you can rename them (see below), but once renamed, the pre-defined name is no longer available.

Variable	Pre-defined name	Comment
v56	VDigIn	This holds the last bit pattern read from the digital inputs with the DIBxx or DIGIN instructions.
v57-v64	VDAC0-VDAC7	These hold the last value written by the sequencer to DACs 0-7. These variables are not updated by arbitrary waveform output so if you use it, you cannot rely on VDACn for the current DAC value.
v61-v64		Also used to emulate the counters for the deprecated LDCNDn and DBNZn instructions used in previous versions of Spike2.
v255	VTick0L	This holds the first (lower) 32-bits of the zero time used by the TICKS instruction and set by the TICK0 instruction. This is an unsigned 32-bit number, which makes it tricky to use in comparisons.
v256	VTick0H	This holds the second (upper) 32-bits of the zero time used by TICKS and set by TICK0.

Variables v57 through v64 hold the last value written by the sequencer to DACs 0 through 7 (and can also be referred to as VDAC0 through VDAC7), v56 holds the last bit pattern read from the digital inputs with the DIBxx or DIGIN instructions, v61 through v64 are also used to emulate the V255 and V256 store the zero time used by the TICKS instruction and are set by the TICK0 instruction.

VAR directive

You can assign each variable a name and an initial value with the VAR directive. Names must be assigned before they are used, usually at the start of the sequence. Each variable can be assigned a name once only, and the name must be unique and not the same as a label, a constant, an expression or an instruction. The syntax is:

```
VAR      Vn, name=expr      ; comment
```

VAR does not generate any instructions. It makes the symbol name equivalent to variable vn and sets the initial value when the sequence is loaded. Anywhere in the remainder of the sequence where vn is acceptable, name can be used. name can be up to 8 characters, must start with an alphabetic character and can contain alphabetic characters and the digits 0 to 9. Variable names are not case sensitive. A variable name must not be the same as an instruction code, a constant or a label.

There is no need to specify a name or an initial value. If no initial value is set, a variable is initialised to 0 even

if not included in a VAR statement. Spike2 automatically assigns V56 the name VDigiIn and variables V57 through V64 the names VDAC0 through VDAC7. If no symbolic name is set, any pre-defined name for the variable is cleared. The following are all acceptable examples:

```
VAR    V1,Wait1=ms(100)    ;Set name and initial value
VAR    V2,UseMe            ;Set name only, so value is 0
VAR    V3=200             ;No name, initialise to a value
VAR    V4                 ;No name, initialised to 0
VAR    V56                ;No name, the pre-defined VDigiIn is cleared
```

When a variable is used in place of a bit pattern in a digital input or output instruction, bits 15 to 8 and bits 7 to 0 have different uses. In the expressions that describe these operations we write $V_n(7-0)$ and $V_n(15-8)$ to describe which bits are used. BAND means bitwise binary AND (if both bits are 1, the output is 1, otherwise 0), BXOR means bitwise exclusive OR (if both bits are different the output is 1, otherwise 0)

When a variable is used in place of a voltage value in a DAC or ADDAC output instruction, the full 32-bit range of the sequencer variable value corresponds to the full range of voltages that can be generated by the DAC, so -2147483648 corresponds to the lowest possible output voltage (-5 or -10 volts), 0 corresponds to a 0 output and 2147483647 corresponds to the highest possible output voltage.

When used in the DACn or ADDACn instructions, only the lower 16 bits of each value are used, so the value -32768 (0x8000) corresponds to the lowest possible output, 0 corresponds to a 0 output and 32767 (0x7fff) corresponds to the highest output. Adding multiples of 65536 (0x10000) to the variable value has no effect on the output.

When used in one of the cosine output angle instructions, the 32-bit variable range from 2147483648 to 2147483647 represents -180 up to +180 degrees. The varValue script in the Scripts folder calculates variable values for the digital and the cosine instructions.

Prior to version 7, you could change the name of a variable part-way through a sequence. From the point of the change, the old name was hidden. This now generates a multiply defined error.

Script access to variables

Scripts can set and read the variable values with the `SampleSeqVar()` script command. You can set initial values from the script as long as you set the values after you create the new data file, but before you start sampling. Values set in this way take precedence over values set by the VAR directive.

Constants

It is often useful to define a numeric value as a named constant. For example, when referencing a table value [V1+Slope] is easier to understand than [V1+23]. It also helps to maintain code; if you need to change a numeric value that is used often make it a named constant and just change the constant once. The = directive does not generate any instructions, it just makes a name equivalent to a number, and the name can be used anywhere a numeric expression is valid. The syntax is:

```
name = expr          ;comment
```

The name must start with an alphabetic character and can contain alphabetic characters and the digits 0 to 9. The name can be up to 8 characters long. Numeric constant names are not case sensitive and must not clash with label, instruction, variable or built-in function names. Examples of use:

```
Wait1 = ms(100)
Wait2 = Wait1*1.3
      DELAY Wait1-1 ;Use constant in an instruction
```

The expression values are calculated and stored as floating point numbers. If they are used in a context that requires an integer value, the fractional part of the number is ignored.

The = directive was added at version 7.00.

The SET SCLK and SDAC directives

These directives control the interval between sequencer clock ticks and the DAC output units. Directives are not part of the sequence and do not occupy a step. These directives should occur before any step-generating code. The SCLK and SDAC directives were added at version 6.03 to separate setting the tick rate from setting

the DAC output units.

```
SET      msPerStep, DACscale, DACoffset
SCLK    msPerStep
SDAC    DACscale, DACoffset
```

The `msPerStep` field sets the milliseconds per step in the range 0.010 to 3000. The table shows the minimum step times and timing resolution for each type of 1401. To ensure accurate timing, `msPerStep` must be an integral multiple of `Resolution` or Spike2 will not sample. This is checked each time Spike2 samples data.

	Power	Micro1401
Minimum step (ms)	.010	.010
Resolution (ms)	.001	.001

If there is no `SET` or `SCLK` directive, the sequence runs at the default rate, which is 10 milliseconds per step unless the script command `SampleSeqClock()` has changed it. If there is no `SET` or `SDAC` directive, `DACScale` is 1 and `DACOffset` is 0 for a ± 5 Volt system and `DACScale` is 2 for a ± 10 Volts system. The sequencer expression functions `s()`, `ms()`, `us()`, `Hz()` and `VHz()` use the `msPerStep` value. If you use these functions, the `SET` or `SCLK` directive must occur first in the sequence.

DAC scaling

The DACs in the 1401 are implemented so that the full range maps onto the full range of 16-bit signed integer numbers (-32768 to +32767). If you have 16-bit DACs a change of 1 changes the output by the smallest step possible. With 12-bit DACs (Micro1401 mk II), the smallest step is a change of 16. For most purposes, it is easier to work in units such as Volts or millivolts rather than these DAC units. However, the 1401 works in DAC units and if you set DAC values using variables, the variable values are based on DAC units. In the `SET` and `SDAC` directives, the `DACscale` and `DACoffset` fields define the conversion from user units into DAC units. The standard values of 1.0 for the scale and 0.0 for the offset make the full scale DAC values run from -5 to +4.99985. As most systems have ± 5 Volt analogue systems, the standard scale and offset let you work with the DACs in Volts.

`DACScale` The number of user units that correspond to an output change of 6553.6 DAC units (1 Volt for a ± 5 Volt system, 2 Volts for a ± 10 Volt system). The standard value 1.0 is used if you omit `DACScale`.

`DACOffset` The user units that correspond to 0 DAC units (0 Volt output). The standard value 0.0 is used if you omit `DACOffset`.

Please remember that `DACScale` and `DACOffset` do not change the DAC outputs in any way. They are a convenience to allow you to enter values in units that are appropriate to your application. The sequencer expression functions `VDAC16()` and `VDAC32()` use `DACScale` and `DACOffset`, so they must come after the `SET` or `SDAC` directives.

For example, consider the case where the DACs drive a patch clamp amplifier where a change of 2.5 Volts into the amplifier causes a 200 mV potential at the cell and 0 Volts into the amplifier is 0 mV at the cell. For a ± 5 Volt system, 2.5 Volts is 16384 DAC units, so `DACScale` is $200 * 6553.6 / 16384$, which is 80. `DACOffset` is 0, as an output of 0 produces 0 mV at the cell.

If you have both ± 5 and ± 10 Volt systems, and you want to specify the output in Volts, you can use:

```
SDAC    DRange() / 5, 0
```

which will set the `DACScale` value to 1.0 on a ± 5 Volt system and to 2.0 on a ± 10 Volts system.

Table of values

From Spike2 version 5.06 onwards the sequencer supports a table of 32-bit values.

Declaring the table

You declare that a table exists with the `TABSZ` directive, which normally occurs at the start of your sequence:

```
TABSZ    expr
```

Where `expr` is an expression that sets the number of items in the table. The table size must evaluate to a number in the range 1 to 1000000. Each table item is a 32-bit integer and uses 4 bytes of 1401 memory. The

maximum size in a 1401 with 1MB of memory, and assuming that there is no arbitrary waveform output, is around 150000 items. The first table item has an index number of 0, the second item is index 1, and so on.

Setting table data

From the script language you can move data between an integer array and the table with the `SampleSeqTable()` function. You can also preset table data from the sequence with the `TABDAT` directive, which must come after the `TABSZ` directive:

```
TABDAT expr
TABDAT expr,expr,expr...
```

Where `expr` is an expression that evaluates to a 32-bit integer. Each `TABDAT` directive adds data to the table, starting at the beginning. The sequencer compiler will flag an error if you define more data that will fit in the table. Table data declared in this way is stored separately from the sequence and is transferred to the 1401 when you create a new data file to sample. If you do not set the table data with the `TABDAT` directive or from a script, the values in the table are undefined.. The `TabPos()` expression has the value of the number of data items that have been defined at the point where it is used. For example:

```
TABDAT TabPos() ;a table item that holds its own index.
```

Accessing table data

Although you can move data between one of the variables and the table with the `TABLD` and `TABST` instructions, many instructions access the table directly. It takes more time to use a table than to use a variable.

All references to the table use the contents of one of the variables as an index into the table plus an optional offset as: `[Vn]` or `[Vn+off]` or `[Vn-off]`. The offset `off` is an expression that evaluates to a number in the range `-1000000` to `1000000`. For example, if `v1` holds 10, `[v1]` refers to the contents of index 10, `[v1-10]` refers to index 0 and `[v1+10]` refers to index 20. Out of range table indices read 0 and are non-destructive.

The `TABINC` instruction makes it easy to increment a variable used as a table index and branch until the increment generates an index outside the table. The following example generates five DAC outputs at 5 different intervals:

```

SET      1,1,0           ; lms per step, normal scales
TABSZ    10              ; table of 10 items
oMs      = TabPos()      ; offset to milliseconds
TABDAT   ms(1000)-3     ; 1000 ms in sequencer steps-3
oVolt    = TabPos()      ; offset to Volts
TABDAT   VDac32(1)      ; 1 Volt
oNext    = TabPos()      ; offset to next set of entries
TABDAT   ms(100)-3,VDac32(2) ; 100 ms, 2 Volts
TABDAT   ms(50)-3,VDac32(3) ; 50 ms 3 Volts
TABDAT   ms(500)-3,VDac32(-1) ; 500 ms -1 Volt
TABDAT   ms(200)-3,VDac32(0) ; 200 ms 0 Volts

TLOOP:   MOVI    V1,0     ; use V1 as table index, set 0
         DELAY   [V1+oMs] ; programmed delay
         DAC     0,[V1+oVolt] ; set DAC 0 to the value
         TABINC  V1,oNext,TLOOP ; add 2 to V1, branch if in table

```

In this case, we could just as easily have set `oMs` to 0, `oVolt` to 1 and `oNext` to 2, or used the values 0, 1 and 2 in the code. However, by using constants, we make it easy to extend the number of items per step. For example, if we decided to insert a new value in the table, making three items per step, we could do so without having to work through our code to check if each occurrence of 0, 1 or 2 was correct or needed changing.

Long data sequences

To output a very long data sequence, you can use the table as a buffer. The basic idea is to divide the table into two halves and use the `SampleSeqTable()` script command to transfer new data into the half of the table that the sequence is not using. To find out where the sequence has reached, look at the value of the variable used as an index with `SampleSeqVar()`. Set a large enough table size so that the time taken to use half the table is several seconds.

Including files

There are times when you will want to reuse definitions or code in multiple projects. You could do this with copy and paste, but it can be more convenient to use `#include` to include files into a sequence. A file that is included can also include further files. We call these nested include files. Only the first `#include` of a file has any effect. A subsequent `#include` of the same file is ignored. This prevents problems with files that include each other and stops multiple definitions when two files include a common file. A `#include` command must be the first non-white space item on a line. There are two forms of the command:

```
#include "filename" ;optional comment
#include <filename> ;optional comment
```

where `filename` is either an absolute path name (starting with a `\` or `/` or containing a `:`), for example `C:\Sequences\MyInclude.pls`, or is a relative path name, for example `include.pls`. The difference between the two command forms lies in how relative path names are treated. The search order for the first form starts at item 1 in the following list. The search for the second form starts at item 3.

1. The folder where the file with the `#include` command lives. If this fails...
2. The folder of the file that included that file up to the top of the list of nested include files. If this fails...
3. Any `\include` folder in the `Spike8` folder inside your `My Documents` folder. If this fails...
4. Any `\include` folder in `\Users\Public\Public Documents\Spike8Shared`. If this fails...
5. Any `\include` folder in the folder in which `Spike2` is installed. If this fails...
6. Search the current folder.

Included files are always read from disk, even if they are already open. If you have set the `Edit menu Preferences` option to save modified scripts and sequences before running, modified include files are automatically saved when you compile. If this option is not set, the output sequencer compiler will stop with an error if it finds a modified include file. You must save the included file to compile your sequence.

There are no restrictions on what can be in an included file. However, they normally contain constant and variable definitions and possibly user-defined code. It is usually a good idea to put all `#include` commands at the start of a sequence file so that anyone reading the source is aware of the scope of the sequence.

The `#include` command for output sequences was added to `Spike2` at version 7.00 and is not recognised by any version before this. A typical file using `#include` might start with:

```
#include <sysinc.pls> ;my system specific includes
#include "include/proginc.pls" ;search relative to source folder
set 1,1,0 ;start of my code...
```

Opening included files

If you right-click on a `#include` command line, and `Spike2` can locate the included file, the context menu will hold an entry to open it. The search for the file follows that described above, except that it omits step 2.

Errors in included files

If an error is detected during the compilation of an included file, an error message is displayed at the top of the original window indicating which included file has a problem, and the included file is opened (if it can be found) and the offending line is highlighted.

Changing the sequence during sampling

As long as sampling starts with a text or graphical sequence running, it can be replaced during sampling. The `Sampling menu Change Output Sequence` command lets you choose a text sequence file on disk and the `Current` button in the `Sequence editor` sets the current document as the sequence. When sampling starts, `Spike2` allocates sufficient space in the 1401 for the first sequence and for any table space it uses. If you want to load sequences during sampling that are larger or needs more table space you must reserve space using the `Sequencer` tab of the `Sampling configuration dialog`.

When a new sequence loads, all DAC values, digital output values, variable and table values are preserved except variables that are given initial values by the `VAR` directive and table values that are given values with the `TABDAT` directive.

Sequencer instruction reference

Each instruction is followed by an example. The examples show a preferred format, however the system is flexible. For example, a comma should separate arguments, but a space is also accepted. The patterns used for digital ports should be enclosed by square brackets, however you may omit the brackets if you wish.

Many of these instructions allow you to use a variable or a table entry in place of an argument. In this case, the alternatives are separate by a vertical bar, for example:

```
DELAY    expr | Vn | [Vn+off] , OptLB
```

This means that the first argument can be an expression, a variable or a table entry. There is no explicit documentation for the use of the table, except in `TABLD` and `TABST`. Where table use is allowed it is written as `[Vn+off]`. If you use a table value in an instruction, the effect is exactly the same as using a variable with the same value as the table entry.

Digital I O

These instructions give you control over the digital output bits and allow you to read and test the state of digital input bits 7-0.

DIGOUT	Write to digital output bits 15-8
DIGLOW	Write to digital output bits 7-0
DIBNE, DIBEQ	Read digital input bits 7-0, copy to V56 test and branch
DISBNE, DISBEQ	Test last read digital inputs (in V56) and branch
WAIT	Wait for a pattern on the lower 8 digital inputs
(DIGIN)	Read and test digital inputs
(BZERO, BNZERO)	Branch as result of DIGIN test

DIGOUT

The `DIGOUT` instruction changes the state of digital output bits 15-8. The output changes occur at the next tick of the output sequencer clock, that is they are precisely timed (unlike `DIGLOW`).

```
DIGOUT [pattern] | Vn | [Vn+off] , OptLab
```

pattern This determines the new output state. You can set, reset or invert each output bit, or leave a bit in the previous state. The pattern is 8 characters long, one for each bit, with bit 15 at the left and bit 8 at the right. The characters can be “0”, “1”, “i” or “.” standing for *clear*, *set*, *invert* or *leave alone*. You may omit the square brackets, however the `Format` command will insert them.

```
DIGOUT [...001i] ;clear bits 3 and 2, set 1, invert 0
DIGOUT [.....i] ;invert 0 again to produce a pulse
DIGOUT V10      ;use variable V10 to set the pattern
```

Vn With a variable the new output is: (old output BAND Vn(7-0)) BXOR Vn(15-8). The variable equivalent of `[...001i]` is `241+256*3`, and of `[.....i]` is `255+256*1`. If you use a table value, set the same value in the table that you would use for a variable. You can use the `VarValue` script in the `Scripts` folder to calculate variable or table values.

OptLB If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example produces ten 1 millisecond pulses 100 milliseconds apart.

```

      SET      1          ;run at 1 ms per step
      MOVI    V1,10      ;V1 holds the number of pulses
LOOP:  DIGOUT  [.....1]   ;bit 0 high      >Pulsing
      DIGOUT  [.....0]   ;bit 0 low      >Pulsing
      DELAY   ms(100)-4  ;4 inst in the loop >Pulsing
      DBNZ   V1,LOOP     ;count down    >Pulsing
      HALT                   ;finished    >Done
```

DIGLOW

DIGLOW changes the state of the 1401 digital output bits 7-0. Unlike DIGOUT, the output changes occur immediately, they do not wait for the next sequencer clock tick. You can take advantage of this to change all 16 digital outputs almost simultaneously (within a few microseconds) by using DIGOUT followed by DIGLOW.

	DIGLOW	[pattern]/Vn [Vn+off],OptLB
pattern	This determines the new output state. The pattern is 8 characters long, one for each bit, with bit 7 at the left and bit 0 at the right. The characters can be “0”, “1”, “i” or “.” standing for <i>clear</i> , <i>set</i> , <i>invert</i> or <i>leave alone</i> . You may omit the square brackets, however Format will insert them.	
	DIGLOW	[...001i] ;clear bits 3 and 2, set 1, invert 0
	DIGLOW	[.....i] ;invert 0 again to produce a pulse
	DIGLOW	V10 ;use variable V10 to set the pattern
Vn	With a variable the new output is: (old output BAND Vn(7-0)) BXOR Vn(15-8). The variable equivalent of [...001i] is 241+256*3, and of [.....i] is 255+256*1. If you use a table value, set the same value in the table that you would use for a variable. You can use the VarValue script in the Scripts folder to calculate variable or table values.	
OptLB	If this label is present it sets the next instruction, otherwise the next sequential instruction runs.	

DIBEQ DIBNE

These instructions test digital input bits 7-0 against a pattern. DIBEQ branches on a match. DIBNE branches on no match. Both instructions copy digital input bits 7-0 to V56 (VDigIn), for use by DISBEQ and DISBNE.

	DIBNE	[pattern] Vn [Vn+off],LB
	DIBEQ	[pattern] Vn [Vn+off],LB
pattern	This is 8 characters, one for each input bit. The characters can be “0”, “1” and “.” meaning match 0 (TTL low), match 1 (TTL high) or match anything. The bit order in the pattern is [76543210]. You may omit the square brackets, however the Format command inserts them.	
Vn	With a variable (input BAND Vn(7-0)) BXOR Vn(15-8) is 0 for a match, else is not a match.	
LB	The branch destination label if the input is a match (DIBEQ) or is not a match (DIBNE).	

This example waits for a pulse sequence in which the falling edges of two consecutive pulses are less than 2*v1+2 sequencer clock ticks apart. It waits for a falling edge, waits for a rising edge with a time out and then waits for the next falling edge with a time out. If timed out, we start again. If the input signal has high states less than three ticks wide, or low states less than 2 ticks wide, this example may miss them.

WHI:	DIBNE	[.....1],WHI	;wait until high	>Wait high
SETTO:	MOVI	V1,24	;set 50 step timeout	>Wait low
WLO:	DIBNE	[.....0],WLO	;wait for falling	>Wait low
TOHI:	DIBEQ	[.....1],TOLO	;wait for high	>Wait high
	DBNZ	V1,TOHI	;loop if not timed out	>Wait high
	JUMP	WHI	;timed out, restart	>Restart
TOLO:	DIBEQ	[.....0],GOTIT	;jump if found events	>Wait low
	DBNZ	V1,TOLO	;loop if not timed out	>Wait low
	JUMP	SETTO	;timed out, restart	>Restart
GOTIT:	...		;here for 2 close pulses	

DISBEQ DISBNE

These instructions test digital input bits 7-0 read by the last DIBEQ, DIBNE or WAIT against a pattern. DISBEQ branches on a match. DISBNE branches if it does not match.

	DISBNE	[pattern] Vn [Vn+off],LB
	DISBEQ	[pattern] Vn [Vn+off],LB
pattern	This is 8 characters, one for each input bit. The characters can be “0”, “1” and “.” meaning match 0 (TTL low), match 1 (TTL high) or match anything. The bit order in the pattern is [76543210]. You may omit the square brackets, however the Format command inserts them.	

- Vn** With a variable (input BAND Vn(7-0)) BXOR Vn(15-8) is 0 for a match, else is not a match.
- LB** The branch destination label if the input is a match (DISBEQ) or not a match (DISBNE).

This example shows a typical use of this instruction. We run trials signalled by external equipment that writes the trial type to digital input bits 1 and 0; 00 means no trial, 01, 10 and 11 select trial types 1, 2 and 3.

```
TRWAIT: 'W DIBEQ [.....00],TRWAIT ;wait for trial >wait...
          DISBEQ [.....01],TRIAL1
          DISBEQ [.....10],TRIAL2
          DISBEQ [.....11],TRIAL3
```

WAIT

The **WAIT** instruction causes the sequence to wait until bits 7-0 of the 1401 digital input match a pattern. The digital input port is sampled once every sequencer clock tick until the pattern is found, or until the sequence is sent elsewhere by a keyboard command. **WAIT** copies digital input bits 7-0 to V56 (VDigIn) for use by **DISBEQ** and **DISBNE**. It is usually a good idea to have a display message explaining what you are waiting for.

```
WAIT [pattern]|Vn|[Vn+off]
```

pattern This is the input condition to match before the sequence can continue. It is 8 characters long, one for each input bit. The characters can be “1”, “0” or “.” indicating that the input bit in that position must be a one, a zero or don't care. The bits are given in the order [76543210]. You may omit the square brackets round the pattern if you wish; the **Format** command will insert them.

```
WAIT [.....1] ;wait for bit 0 set>Wait for bit 0
WAIT [0.....0] ;wait for 7 and 0 clear>Wait 7&0 low
```

- Vn** With a variable, (input BAND Vn(7-0)) BXOR Vn(15-8) must be 0 to continue.

This instruction is shorthand for:

```
HERE: DIBNE [pattern]|Vn|[Vn+off],HERE
```

DIGIN BZERO BNZERO

These obsolete instructions are provided for backwards compatibility. **DIBNE** and **DIBEQ** are more efficient. **DIGIN** reads digital input bits 7-0, compares them with a pattern and saves the result. The result can be tested with the branching instructions **BZERO** and **BNZERO**. The comparison result is saved until the next **DIGIN**.

```
DIGIN [pattern]|Vn|[Vn+off] ;Test input state
BZERO LB ;branch to LB if result is zero
BNZERO LB ;branch to LB if result non-zero
```

pattern This is 8 characters, one for each input bit in the order [76543210]. The characters can be “.”, “0”, “1” and “c”. For “0” or “1”, the result for that bit is 0 if the input bit was the same, or 1 if it was different. “c” means copy the input bit to the result (this is the same as “0”). The result for “.” is always zero. You may omit the square brackets round the pattern, however the **Format** command will insert them.

- Vn** With a variable the result is (input BAND Vn(7-0)) BXOR Vn(15-8). The variable equivalent of [0.c0110.] is 190 + 256*12. The **VarValue** script calculates values equivalent to patterns.

LB The branch destination if the last **DIGIN** produced a zero (**BZERO**) or non-zero (**BNZERO**) result.

```
Loop: DIGIN [0.c0110.] ;assume input is 01101011
      BZERO GoOn ;result is 00100110 so no branch
      BNZERO Loop ;This will branch
GoOn: ...
```

DAC outputs

The output sequencer supports up to 8 DAC (Digital to Analogue Converter) outputs. The Power1401 has four DACs and the Micro1401 has two. However, the Power1401 can be expanded with up to 8 DAC outputs. The last value written to DACs 0-7 is stored in variables V57-V64 (you can also refer to these variables as VDAC0-

VDAC7). The values are stored as 32-bit numbers with the full 32-bit range corresponding to the full range of the DAC. This high resolution allows us to ramp the DACs smoothly .If you write to a DAC that does not exist in your 1401, the variable associated with the DAC is set as if the DAC were present.

Values written to the DACs are expressed in units of your choice. The SET or SDAC directive determines the conversion between the numbers you supply and the DAC outputs. The standard settings for a system with ±5 Volts DACs is to set the DAC outputs in Volts.

The Power1401 DAC 2 and 3 outputs are on the rear panel 37-way Cannon D type Analogue Expansion connector. DAC 2 is pin 36 and DAC 3 is pin 37. Suitable grounds are on adjacent pins 18 and 19. If you have a Power1401 top-box with additional front panel DACs, the two rear DACs are mapped to the two highest numbered DACs. For example, with a 2709 Spike2 Top Box, DACs 2 and 3 are available as front panel BNC connections, and the rear panel DACs become DAC 4 on pin 36 and DAC 5 on pin 37.

DAC	Set DAC value (for DACs 0-7)
(DACn)	Version 3 compatible, set DAC n (0-3) to a value
ADDAC	Increment DAC by a value
ADDACn	Version 3 compatible, increment DAC n (0-3) by a value
RAMP	Set automatic DAC ramping to a target value

DAC ADDAC

The DAC instruction can write a value to any of the 8 possible DAC outputs. The ADDAC instruction adds a value to the DAC output. The output value changes immediately unless the DAC is in use by the arbitrary waveform output, in which case the result is undefined.

DAC	n, expr	Vn [Vn+off], OptLB
ADDAC	n, expr	Vn [Vn+off], OptLb

n The DAC number, in the range 0-7. Variable 57+n is set to the new DAC value such that the full DAC range spans the full range of the 32-bit variable.

expr The DAC value to write or to add. The value units depend on the SET or SDAC directive; the standard units are Volts. It is an error to give a value that exceeds the DAC output range.

Vn When a variable is used, the full range of the 32-bit variable corresponds to the full range of the DAC. You can use the VDACC32() function to load a variable using user-defined DAC units.

OptLB If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example sets DAC 2 to 0 Volts, then ramps it to 4.99 Volts in 1 second using steps of 0.01 Volts. The example also shows how to do the same ramp using variables. You can also use RAMP to ramp a DAC.

	SET	1,1,0	;1 ms per step, DAC scaled to Volts
'R	DAC	2,0	;Ramp 0 to 5 >Ramping
	MOVI	V1,499	;499 steps >Ramping
RAMP1:	ADDAC	2,0.01	;0.01V increment >Ramping
	DBNZ	V1,RAMP1	;count increments >Ramping
	HALT		;task finished >Done
'V	MOVI	V3,VDACC32(0)	;Use variables >Ramping
	MOVI	V2,VDACC32(0.01)	;increment in V2 >Ramping
	MOVI	V1,499	;499 steps >Ramping
	DAC	2,V3	;set initial value>Ramping
RAMP2:	ADDAC	2,V2	;add increment >Ramping
	DBNZ	V1,RAMP2	;count increments >Ramping
	HALT		;task finished >Done

It is a property of the signed integer numbers we use that if you add 1 to the maximum possible positive number, the result is the minimum possible negative number. If you use ADDAC repeatedly to add the same value, eventually you will run off the end of the DAC range and come back in at the other end.

Physical DAC units run from -32768 to +32767. In a ±5 Volt system with 16-bit DACs, this is -5.0000 to +4.99985 Volts. The DAC unit value for +5 Volts is +32768, but this number does not exist in 16-bit signed integers and wraps around to -32768. Because it often happens that users want to set the DAC to full scale, for the DAC command used with expr (not with Vn), we change requests to set +32768 units to set -32767 units. When used with a variable (Vn), the top 16 bits of the 32-bit variable are physical DAC units, the lower 16 bits provide extra accuracy when ramping the DACs.

Unlike the digital outputs, the DAC output changes when the instruction runs, not at the next sequencer clock tick. This means that the changes may have a time jitter of a few microseconds.

DACn ADDACn

These commands provide backwards compatibility with old Spike2 versions and should be avoided in new sequences. The `DACn` and `ADDACn` commands (with `n = 0, 1, 2` or `3`) set and change the 1401 DAC outputs. `expr` is the new DAC output level, or change in level. Variable `57+n` is set to the new DAC value such that the full DAC range spans the full range of the 32-bit variable. These commands do not support table use.

	<code>DACn</code>	<code>expr</code>	<code>Vn,OptLB</code>
	<code>ADDACn</code>	<code>expr</code>	<code>Vn,OptLB</code>

`expr` The value to assign to DAC `n` (`DACn` instruction) or to add to the output level (`ADDACn` instruction). The units of the DAC values are usually Volts, but can be changed by setting a scale factor with `SET` or `SDAC`.

`Vn` variable values `-32768` to `32767` correspond to the full DAC range. You can use `VDAC16()` to load a variable using user-defined DAC units. You can also use the `VarValue` script to calculate variable values. The value saved in `V57+n` is `Vn*65536`.

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

The important difference between these commands and `DAC` and `ADDAC` is where a variable is used. The bottom 16-bits of the variable are written to the DAC. In the case of the `DAC` and `ADDAC` commands, the upper 16 bits of the variable are written to the DAC. This command is sometimes useful with the `CHAN` command when reading a waveform or a DAC channel, as it reads values in the range `-32868` to `32767`.

RAMP

This command starts a DAC ramping, with automatic updates on every sequencer step. If the DAC was generating cosine output, the cosine output stops. The DAC ramps from the current value until it reaches a target value, when the DAC cycle flag sets. You can use `WAITC` to test for the end of the ramp. The `RATE` instruction stops a ramp before it reaches the target value.

	<code>RAMP</code>	<code>n, target</code>	<code>Vn, slope</code>	<code>Vs</code>	<code>[Vs+off]</code>
--	-------------------	------------------------	------------------------	-----------------	-----------------------

`n` The DAC number in the range `0-7` for the Power1401 or `0-1` for the Micro1401.

`target` This is the DAC value at which to end the ramp. The units of the DAC values are usually Volts, but can be changed by setting a scale factor with `SET` or `SDAC`.

`Vn` When a variable is used for the target, the full range of the 32-bit variable corresponds to the full range of the DAC. You can use the `VDAC32()` function to load a variable using user-defined DAC units.

`slope` This expression sets the DAC increment per sequencer step. The sign of the value you set here is ignored as the sequencer works out if it must ramp upwards or downwards to achieve the desired target value. If your DAC is calibrated in Volts, to achieve a slope of 1 Volt per second, use `1.0/s(1.0)` for the slope.

`Vs` You can also set the slope from a variable or by reading it from the table. In this case, the full range of the 32-bit value represents the full range of the DAC. The absolute value of the 32-bit value is used to change the DAC on each step. To get a slope of 1 user unit per second, use `VDAC32(1.0)/s(1.0)` as the value.

This example ramps the DAC 1 from its current level to 1 Volt over 3 seconds, waits 1 second, then ramps it down to 0 Volts over 5 seconds.

	<code>RAMP</code>	<code>1,1.0,1.0/S(3)</code>	<code>;start with zero size</code>
	<code>...</code>		<code>;other instructions during ramp</code>
<code>WT1:</code>	<code>WAITC</code>	<code>1,WT1</code>	<code>;wait for ramp to end >Ramp to 1</code>
	<code>DELAY</code>	<code>S(1)-1</code>	<code>;wait for a second >Wait 1 sec</code>
	<code>RAMP</code>	<code>1,0,1.0/S(5)</code>	<code>;ramp down</code>
<code>WT2:</code>	<code>WAITC</code>	<code>1,WT2</code>	<code>;wait for ramp >Ramp to 0</code>

The `OFFSET` command has an example that uses `RAMP` with cosine waves.

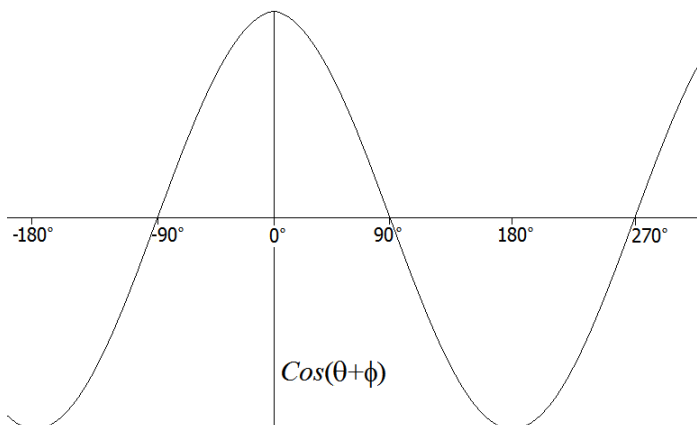
Cosine output control instructions

The sequencer can output cosine waveforms of variable amplitude and frequency through Power1401 DACs 0 to 7 and Micro1401 DACs 0 and 1. Attempts to use Micro1401 DACs 0 and 1 are treated as NOP. When enabled, the cosine value is computed and output every step. The output (in a 5 Volt 1401) is:

$$\text{output in Volts} = 5 A \cos(\theta + \phi) + \text{offset}$$

where: A is an amplitude scaling factor in the range 0 to 1
 θ an angle in the range 0° to 360° that changes each step (set by ANGLE)
 ϕ is a fixed phase in the range -360° to 360° (set by PHASE)
 offset a voltage offset set by the OFFSET instruction

θ changes every step by $\delta\theta$. A cycle of the cosine takes $360/\delta\theta$ steps. You can change the angle increment immediately, or you can delay the change until the next time θ passes through 0° . You can set $\delta\theta$ in the range 0° up to 360° to an accuracy of about 0.0000001° . With the sequencer running at 1 kHz, you can output frequencies up to 500 Hz with a frequency resolution of around 0.00012 Hz. Ideally the output would be passed through a low pass filter with a corner frequency at one half or less of the sequencer step rate to smooth out the steps in the cosine wave.



By adjusting ϕ you control the output cosine phase where θ passes through zero. Unless you set the value (PHASE), it is zero and the zero crossing occurs at the peak of the sinusoid. To have the output rising through 0, set the phase to -90 .

Each time θ passes through zero a *new cycle* flag sets. The RAMP, RATEW, RINCW, WAITC and CLRC instructions clear the flag.

SZ (CSZ, DSZ)	Set the cosine output amplitude
SZINC (CSZINC, DSZINC)	Change the cosine output amplitude
RATE (CRATE, DRATE)	Set the cosine angular increment per step
RATEW (CRATEW, DRATEW)	As RATE, but waits for phase 0
ANGLE (CANGLE, DANGLE)	Set cosine angle for the next step
WAITC (CWAIT, DWAIT)	Branch until cosine phase 0
RINC (CRINC, DRINC)	Change the cosine angle increment per step
RINCW (CRINCW, DRINCW)	As RINC but waits for phase 0
PHASE (CPHASE, DPHASE)	Defines what phase 0 means
OFFSET (COFF, DOFF)	Offset for sinusoidal output
CLRC (CWCLR, DWCLR)	Clear the new cycle flag

Obsolete commands

Before Spike2 version 5.06, the cosine output instructions supported 2 DACs through the Cxxxx and Dxxxx instructions. The Cxxxx family used DAC 1 on the Power1401 and Micro1401 and DAC 3 on the 1401plus. The Dxxxx family used DAC 0 on the Power1401 and Micro1401 and DAC 2 on the 1401plus. The descriptions include the old forms of the commands, marked *obsolete*. The DAC numbers next to obsolete commands show the 1401plus DAC number in brackets. There are no plans to remove the old commands, but new sequences should avoid them.

SZ

This instruction sets the waveform amplitude. If a wave is playing, the amplitude changes at the next sequencer step. The amplitude is set to 1.0 when sampling starts.

SZ n, expr | Vn | [Vn+offset], OptLB ;DAC n

CSZ	expr Vn [Vn+off], OptLB	;DAC 1(3) - Obsolete
DSZ	expr Vn [Vn+off], OptLB	;DAC 0(2) - Obsolete

- n** The DAC number in the range 0-7 (available DACs depend on the 1401 type).
- expr** The cosine amplitude in the range 0 to 1. A cosine with amplitude 1.0 uses the full DAC range.
- Vn** Variable values 0 to 32768 correspond to amplitudes of 0.0 to 1.0; values outside the range 0 to 32768 cause undefined results.
- OptLB** If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

SZINC

These instructions change the waveform amplitude. The change is added to the current amplitude. If the result exceeds 1.0, it is set to 1.0. If it is less than 0, the result is 0.

SZINC	n, expr Vn [Vn+off], OptLB	;DAC n
CSZINC	expr Vn [Vn+off], OptLB	;DAC 1(3) - Obsolete
DSZINC	expr Vn [Vn+off], OptLB	;DAC 0(2) - Obsolete

- n** The DAC number in the range 0-7 (available DACs depend on the 1401 type).
- expr** The change in the waveform scale in the range -1 to 1.
- Vn** A variable value of 32768 is a scale change of 1.0, -16384 is -0.5 and so on.
- OptLB** If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

You can gradually increase or decrease the wave amplitude. For example, the following increases the amplitude from zero to full scale (we assume that the waveform is playing):

	SZ	0,0.0	;start with zero size
	MOVI	V1,100	;proceed in 1% increments
loop:	SZINC	0,0.01	;a 1% increase
	DELAY	ms(100)-2	;show some of the waveform at this size
	DBNZ	V1,loop	;loop 100 times

RATE

This sets the angle increment in degrees per step, which sets the cosine frequency. If the nominated DAC was ramping, this cancels the ramp. You can stop the cosine output with a rate of 0. Any non-zero value starts the cosine output.

RATE	n, expr Vn [Vn+off], OptLB	;DAC n
CRATE	expr Vn [Vn+off], OptLB	;DAC 1(3) - Obsolete
DRATE	expr Vn [Vn+off], OptLB	;DAC 0(2) - Obsolete

- n** The DAC number in the range 0-7 (available DACs depend on the 1401 type).
- expr** The angle increment per step in the range 0.000 up to 180 degrees. The Hz() function calculates the increment required for a frequency.
- Vn** For a variable, the value 11930465 is an increment of 1 degree. The VHz() function can be used to set a variable value equivalent to an angle in degrees.
- OptLB** If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example starts cosine output at 10 Hz, runs for 10 seconds, and then stops it. This is then repeated using a variable to produce the same effect:

	SET	1,1,0	;1 ms per step
'C	RATE	0,HZ(10)	;start output at 10 Hz
	DELAY	S(10)-1	;delay for 10 seconds >Sine wave
X:	'S	RATE	0,0 ;stop output
		HALT	>Stopped
'V	MOVI	V1,VHz(10)	;set V1 equivalent of 10 Hz
	RATE	0,V1	;start at 10 Hz
	DELAY	S(10)-1,X	;delay then goto exit >Sine wave

RATEW

This instruction performs the same function as RATE, except that the change is postponed until the next time Theta passes through 0 degrees. RATEW cannot start output; a sinusoid must already be running to pass phase 0. It can stop output, but does not remove the overhead for using cosine output. This instruction clears the new cycle flag (see WAITC).

```
RATEW      n,expr|Vn|[Vn+off]      ;DAC n
CRATEW     expr|Vn|[Vn+off]      ;DAC 1(3) - Obsolete
DRATEW     expr|Vn|[Vn+off]      ;DAC 0(2) - Obsolete
```

- n** The DAC number in the range 0-7 (available DACs depend on the 1401 type).
- expr** The angle increment in the range 0.000 to 180 degrees. The Hz() built-in function calculates the increment required for a frequency.
- Vn** For a variable, the value 11930465 is an increment of 1 degree. The vHz() function can be used to set a variable value equivalent to an angle in degrees.
- OptLB** If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example starts cosine output at 10 Hz, runs for 1 cycle, changes to 11 Hz for one cycle, then stops:

```
SET 1,1,0      ;1 ms per step
ANGLE 0,0      ;make sure we are at phase 0
RATE 0,HZ(10)  ;start output at 10 Hz
RATEW 0,HZ(11) ;request 11 Hz next time around
CYCLE10: WAITC 0,CYCLE10 ;wait for the cycle>10Hz
CYCLE11: WAITC 0,CYCLE11 ;wait for the cycle>11Hz
RATE 0,0      ;stop output
```

ANGLE

This changes the cosine angular position. It takes effect on the next instruction when the angle increment is added to the value set by this instruction and the result is output.

```
ANGLE      n,expr|Vn|[Vn+off],OptLB ;DAC n
CANGLE     expr|Vn|[Vn+off],OptLB   ;DAC 1(3) - Obsolete
DANGLE     expr|Vn|[Vn+off],OptLB   ;DAC 0(2) - Obsolete
```

- n** The DAC number in the range 0-7 (available DACs depend on the 1401 type).
- expr** The phase angle to set in the range -360 up to +360.
- Vn** For a variable, the value 11930465 is a phase of 1 degree (to be precise, 4294967296/360 is a phase of 1 degree). The vAngle() function converts degrees into a suitable value for a variable.
- OptLB** If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example sets the phase angle to -90 degrees directly, and by using a variable. There is no need to use the vAngle() function; we could have set v1 to -1073741824. However, vAngle(-90) is much easier to understand.

```
ANGLE 1,-90      ;set the DAC 1 cosine angle directly
MOVI  V1,vAngle(-90)
ANGLE 1,V1       ;set using a variable
```

PHASE

This changes the relative phase of the cosine output for the next cosine output. A common use is to change the output from a cosine (maximum value at phase zero) to sine (rising through zero at phase zero).

```
PHASE      n,expr|Vn|[Vn+off],OptLB ;DAC n
CPHASE     expr|Vn|[Vn+off],OptLB   ;DAC 1(3) - Obsolete
DPHASE     expr|Vn|[Vn+off],OptLB   ;DAC 0(2) - Obsolete
```

- n** The DAC number in the range 0-7 (available DACs depend on the 1401 type).

- expr** The relative phase angle to set in the range -360 up to +360. The relative phase is set to 0 when sampling starts. Set -90 for sinusoidal output.
- Vn** For a variable, the value 11930465 is a phase of 1 degree (to be precise, 4294967296/360 is a phase of 1 degree). The `vAngle()` function converts degrees into a suitable value for a variable.
- OptLB** If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example plays a 1 Hz sinusoidal output (assuming that the output is not running).

```
PHASE 2,-90 ;set the DAC 2 phase angle directly
ANGLE 2,0 ;prepare to start as a sine wave
RATE 2,HZ(1) ;start the sinusoid
```

OFFSET

This changes the cosine output voltage offset for the next cosine output.

```
OFFSET n,expr|Vn|[Vn+off],OptLB ;DAC n
COFF expr|Vn|[Vn+off],OptLB ;DAC 1(3) - Obsolete
DOFF expr|Vn|[Vn+off],OptLB ;DAC 0(2) - Obsolete
```

- n** The DAC number in the range 0-7 (available DACs depend on the 1401 type).
- expr** The offset value for sinusoidal output. The units of this value depend on the `SET` or `SDAC` directives; the standard units are Volts. It is an error to give a value that exceeds the DAC output range.
- Vn** When a variable is used, the full range of the 32-bit variable corresponds to the full range of the DAC. You can use the `VDAC32()` function to load a variable using user-defined DAC units.
- OptLB** If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example ramps DAC 0 from 0 to 1 Volt, the runs 5 cycles of a sine wave at 1 Hz, and finally ramps the data back to 0 Volts.

```
SET 1 1 0 ;1 millisecond per step
DAC 0,0 ;use DAC 0 for all output
OFFSET 0,1.0 ;set DAC 0 offset
SZ 0,0.2 ;1 V sinusoid
PHASE 0,-90 ;Prepare sinusoid
ANGLE 0,0 ;set start point
RAMP 0,1.0,1.0/s(1) ;ramp to 1 Volt in 1 sec
RAMPUP: WAITC 0,RAMPUP ;wait for ramp >Ramp up
RATE 0,HZ(1) ;start sinusoid
DELAY S(4.9) ;Sinusoid >Sine
RATEW 0,0 ;stop at cycle end
END: WAITC 0,END ;wait for end >Wait end
RATE 0,0 ;stop now
RAMP 0,0.0,1.0/S(1) ;ramp to 0 Volt in 1 sec
RAMPDN: WAITC 0,RAMPDN ;wait >Ramp down
HALT
```

WAITC

Each time the phase angle of a cosine passes through 0°, or a ramp terminates, a new cycle flag sets. There is a separate flag for each DAC. This flag is cleared by `CLRC`, `RATEW`, `RINCW` and when tested by `WAITC`.

```
WAITC n,LB ;DAC n
CWAIT LB ;DAC 1(3) - Obsolete
DWAIT LB ;DAC 0(2) - Obsolete
```

- n** The DAC number in the range 0-7 (available DACs depend on the 1401 type).
- LB** A label to branch to if the new cycle flag is clear, otherwise the sequencer clears it and continues.

This instruction can produce a pulse one step after the start of each waveform cycle. The following sequence outputs 4 cycles of waveform at different rates on DAC 1, and changes the digital outputs for each cycle.

```
SZ 1,1.0 ;make sure full size
ANGLE 1,0.0 ;make sure we start at phase 0
```

```

        RATE 1,1.0 ;1 degree per step to start with
        DIGOUT [00000001] ;so outside world knows
        RATEW 1,1.2 ;next cycle faster, clear cycle flag
w1:    WAITC 1,w1 ;wait for cycle >1 degree cycle
        DIGOUT [00000010] ;announce another cycle
        RATEW 1,1.4 ;next cycle a bit faster
w2:    WAITC 1,w2 ;wait for cycle >1.2 degree cycle
        DIGOUT [00000011] ;yet another one
        RATEW 1,1.6 ;last cycle a bit faster
w3:    WAITC 1,w3 ;wait for cycle >1.4 degree cycle
        DIGOUT [00000100] ;last cycle number
w4:    WAITC 1,w4 ;wait for end >1.6 degree cycle
        RATE 1,0.0 ;stop waveform
    
```

RINC RINCW

These instructions behave like RATE and RATEW except that they *change* the output rate (angle increment per step) by their argument rather than set it. RINCW clears the new cycle flag.

```

        RINC  n,expr|Vn|[Vn+off],OptLB ;DAC n
        RINCW n,expr|Vn|[Vn+off],OptLB ;DAC n
        CRINC expr|Vn|[Vn+off],OptLB ;DAC 1(3) - Obsolete
        CRINCW expr|Vn|[Vn+off],OptLB ;DAC 1(3) - Obsolete
        DRINC  expr|Vn|[Vn+off],OptLB ;DAC 0(2) - Obsolete
        DRINCW expr|Vn|[Vn+off],OptLB ;DAC 0(2) - Obsolete
    
```

- n The DAC number in the range 0-7 (available DACs depend on the 1401 type).
- expr The change in the angle increment per step. You can use the built-in Hz() function to express the change as a frequency.
- Vn For a variable, the value 11930465 is a change of 1 degree. You can use the VarValue script in the Scripts folder to calculate variable values.
- OptLB If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

This example starts cosine output at 10 Hz and lets you adjust it from the keyboard.

```

        SET 1,1,0 ;1 ms per step
        RATE 1,HZ(10) ;start output at 10 Hz
wt:    JUMP wt ;HALT stops all output>P=+1Hz, M=-1Hz
        'P RINC 1,HZ(1),wt;1 Hz faster
        'M RINC 1,HZ(-1),wt;1 Hz slower
    
```

These instructions can be used to produce waveforms that change gradually in frequency. The following code generates a linear speed increase every two steps on DAC 1:

```

        SZ 1,1.0 ;make sure full size
        ANGLE 1,0.0 ;make sure we start at phase 0
        RATE 1,1.0 ;1 degree per step to start with
        MOVI V1,900 ;in 900 steps of...
loop:  RINC 1,0.01 ;...1/100 degrees to...
        DBNZ V1,loop ;...10 degrees per step
    
```

The next example produces 90 cycles using v10 as a counter, increasing by 0.1 degrees per step per cycle.

```

        SZ 1,1.0 ;make sure full size
        ANGLE 1,0.0 ;make sure we start at phase 0
        RATE 1,1.0 ;1 degree per step to start with
        MOVI V10,90 ;in 90 steps of...
loop:  RINCW 1,0.1 ;...1/10 degrees to...
wait:  WAITC 1,wait ;...(wait for next cycle)...
        DBNZ V10,loop ;...10 degrees per step
    
```

CLRC

This instruction clears the cosine output new cycle flag. If you have been running for several cycles and you want to stop the next time phase 0 is crossed use this instruction immediately before using WAITC.

```

        CLRC n,OptLB ;DAC n
        CWCLR OptLB ;DAC 1(3) - Obsolete
    
```

	DWCLR	OptLB		;DAC 0(2) - Obsolete
n	The DAC number in the range 0-7 (available DACs depend on the 1401 type).			
OptLB	If this label is present it sets the next instruction, otherwise the next sequential instruction runs.			
This example starts a sinusoid and then stops at the next phase 0 crossing after the user requests a stop. Because the sinusoid passes phase 0 in the WAITC instruction and does another step in the RATE 1,0 instruction, we offset the phase by 2 steps. However, this would cause the start of the sinusoid to be 2 steps wrong, so we change the start angle to match.				
	'G	PHASE	1,-2*Hz(2)	; comenstate for ending
		ANGLE	1,2*Hz(2)	; so we start in correct place
		RATE	1,Hz(2)	; 2 Hz output
HERE:	JUMP	HERE		; output is running >Running
'S	CLRC	1		; Stop output
WT:	WAITC	1,WT		; wait for cycle end>Waiting
	RATE	1,0,HERE		; stop and then idle

General control

These instructions do not change any outputs or read data from any inputs. They provide the framework of loops, branches and delays used by the other instructions.

DELAY	Do nothing for a set number of steps
DBNZ	Decrement a variable and branch if not zero
(LDCNTn, DBNZn)	Load counter 1 to 4 (V61-V64), decrement, branch if not zero
Bxx	Compare variables and branch (xx = GT, GE, EQ, LE, LT, NE)
CALL	Branch to a label, save return position
CALLV, (CALLn)	Like CALL, but load a variable (counter 1-4) with a value
RETURN	Branch to instruction after last CALL, CALLV or CALLn
JUMP	Unconditional branch to a label
HALT	Stops the sequencer and waits to be re-routed
NOP	This does nothing for one step (NO OPERATION)

DELAY

The DELAY instruction occupies one clock tick plus the number of extra ticks set by the argument. It produces simple delays of 1 to more than 4,000,000,000 sequencer steps.

	DELAY	expr Vn [Vn+off], OptLB
expr	The extra sequencer clock ticks to delay in the range 0 to 4294967295. The s(), ms() and us() built-in functions convert a delay in seconds, milliseconds or microseconds into sequencer steps.	
Vn	Variable or table index from which to read the number of extra clock ticks.	
OptLB	If this label is present it sets the next instruction, otherwise the next sequential instruction runs.	
This example uses display messages to tell the user what the sequence is doing.		
	SET	1.00,1,0 ;run with 1 millisecond clock ticks
	DELAY	2999 ;wait 2999+1 milliseconds>3 second delay
	DELAY	s(3)-1 ;3 seconds -1 tick delay >3 second delay
	DELAY	V1,LB ;wait V1+1 ms, branch >variable delay
	DELAY	[V1+9] ;V1+9 is table index >table delay

DBNZ

DBNZ (Decrement and Branch if Not Zero) subtracts 1 from a variable and branches to a label unless the counter is zero. It is used for building loops.

	DBNZ	Vn, LB
Vn	The variable to decrement and test for zero.	
LB	Instruction to go to next if the result of the decrement is not zero.	

DBNZ is often used with MOVI to set up loops, for example:

```

      MOVI    V2,1000      ;set times to loop
WT:   DIGOUT  [00000000] ;set all digital outputs low
      DIGOUT  [11111111] ;set them all high
      DBNZ   V2,WT       ;loop 1000 times

```

LDCNTn DBNZn

These obsolete instructions use variables V61 to V64 as counters 1 to 4. New sequences should use MOVI and DBNZ, which can use any variable. LDCNTn loads a counters. DBNZn (Decrement and Branch if Not zero) decrements a counter and branches while the counter is not zero. V61-V64 also store values for DACs 4-7.

```

LDCNTn  expr | Vn
DBNZn   LB

```

n The counter to load or decrement in the range 1 to 4, using variables V61 to V64.

expr The 32-bit integer value to load into the counter.

Vn When a variable is used the counter is set to the 32-bit variable.

LB Instruction to go to next if the result of the decrement is not zero

CALL CALLV CALLn RETURN

These instructions run a labelled part of a sequence and return. CALL, CALLV and CALLn save the next step number to a *return* stack and jump to the labelled instruction. The RETURN instruction removes the top step number from the return stack and jumps to it. CALLV also sets a variable to a constant. CALLn (n=1 to 4) is obsolete and sets the value of the variables that emulate the four counters, V61 to V64. These four variables are also used to store the outputs set for DACs 4-7.

```

CALL    LB           ; use LB as a subroutine
CALLV   LB,Vn,expr   ; Vn = expr, then call LB
CALLn   LB,expr      ; V32+n = expr (n=1, 2, 3 or 4)
RETURN  ; return to step after last CALL

```

LB The next instruction to run. The CALLED section should end with a RETURN.

Vn CALLV copies the value of *expr* to this variable. The obsolete instructions CALL1 sets V61, CALL2 sets V62, CALL3 sets V63 and CALL4 sets V64.

expr A 32-bit integer constant that is copied to a variable. For CALLn only, if *expr* is 0, the variable is set to 256 to be compatible with previous versions.

You can use CALL inside a CALLED subroutine. This is known as a *nested* CALL. If you call a subroutine from inside itself, this is known as a *recursive* CALL. The return stack has room for 64 return addresses. If you use more than this, the oldest return address is overwritten, so your sequence will not behave as you expect.

This example generates different pulse widths from DAC 0. The sequence is written to be independent of the sequencer rate (it must be high enough so that the widths are possible). In this case the rate is set to 5 kHz. The example sets DAC 0 to zero, then pulses for 20 milliseconds twice, once using CALL and once using CALLV. Then after a delay, there is a 50 millisecond pulse.

```

      SET    0.2,1,0      ; Run at 5 kHz, normal DAC scale
      DAC   0,0          ; make sure DAC0 is zero
      MOVI  V3,ms(20)-2  ; these two instructions...
      CALL  PUL          ; ...have the same effect as...
      CALLV PUL,V3,ms(20)-2; ...this one. 20 ms pulse
      DELAY s(1)-1      ; wait 1 second, then...
      CALLV PUL,V3,ms(50)-2; ...a 50 ms pulse
      HALT ; So we don't fall into PUL routine
PUL:  DAC   0,1          ; set DAC value
      DELAY V3          ; wait for time set
      DAC   0,0          ; set DAC back to zero
      RETURN ; back to the caller

```

CALL/CALLV and RETURN let you reuse a block of instructions. This can make sequences much easier to understand and maintain. The disadvantage is the additional steps for the CALL and RETURN. If you need to set

a variable, use `CALLV` and there is only the overhead of the `RETURN` instruction.

JUMP

The `JUMP` instruction transfers control unconditionally to the instruction at the label. Many instructions allow the use of an optional label to set the next instruction, so you can often avoid the need for this instruction. You can also jump using the contents of a register as the destination, or relative to a label (`LB`):

```
JUMP  LB           ; Jump to label
JUMP  (Vn),OptLB  ; use MOVI Vn,LB to set target
JUMP  LB(Vn),OptLB ; Jump to instruction LB+Vn
```

(Vn) The value of variable `Vn` sets the instruction number to jump to.

LB(Vn) Jump to the instruction given by label `LB` plus the contents of `Vn`.

OptLB An optional label to jump to if (Vn) or LB(Vn) is not an instruction number. The first instruction is 0, the last depends on the size of the sequence.

State machine

You can use the `JUMP` instruction to implement a state machine. A state machine is characterised by activities in each state, and transitions between states. State machines are a convenient way to tidy up a complicated sequence of operations involving conditions such as the level of an input signal. For example:

State	Activity	Transition
0	Wait for digital input 0 to be low	To state 1
1	Wait for digital input 0 to go high	To state 2 on high
2	Wait for digital input 0 to go low	To state 3 on low, digital output high on change
3	Wait for 10 seconds	To state 0, digital output low on change

This could be coded as:

```
SET      1.000 1 0
VAR      V1,State=State0 ;initial state
VAR      V2,Until=0      ;Used to time state 3
VAR      V3,Now

IDLE:    ...                ;background actions >=
         JUMP  (State)      ;run state machine >=

STATE0:  DIBNE  [.....0],IDLE ;wait for low      >State 0
         MOVI  State,State1,IDLE ;move on      >"
STATE1:  DIBNE  [.....1],IDLE ;wait for high    >State 1
         MOVI  State,State2,IDLE ;              >"
STATE2:  DIBNE  [.....0],IDLE ;wait for low again >State 2
         DIGOUT [.....1]      ;Digital out high  >"
         TICKS  Until,sTick(1) ;1 second ahead  >"
         MOVI  State,State3,IDLE ;              >"
STATE3:  TICKS  Now,0         ;get current time  >State 3
         BLT   Now,Until,IDLE ;wait for 1 second >"
         DIGOUT [.....0]      ;Digital out low  >"
         MOVI  State,State0,IDLE
```

The `...` at label `IDLE` stands for instructions that you want to run in the background while the state machine runs. Of course, the more background instructions you include, the less frequently the state machine checks the state of the inputs.

HALT

The `HALT` instruction stops the output sequence and removes all overhead associated with it. It does not stop the sequencer clock, which continues to run. Any cosine output will stop, but will restart when the sequence restarts. To restart the sequencer, press a key associated with a sequence step or click a key in the sequencer

control panel. If you associate a display string with this instruction, it appears in the sequencer control panel.

```
HALT >Press X when ready
```

NOP

The NOP instruction (NO OPERATION) does nothing except use up one sequencer clock tick. It can be thought of as the equivalent of DELAY 0.

Variable arithmetic

These instructions perform basic mathematical functions while a sequence runs. You can also compare variables and branch on the result.

ABS	Set a variable to the absolute value of another
ADD	Add one variable to another
ADDI, (ADDIL)	Add a constant value to a variable
MOV	Copy one variable to another
MOVI, (MOVIL)	Move a constant value into a variable
MUL, MULI	Multiply two variables, multiply by a constant
NEG	Move minus the value of a variable to another
SUB	Subtract one variable from another
DIV, RECIP	Division and reciprocal of variables

Compare variable

These instructions compare a variable with a variable or a 32-bit expression or a table entry and branch on the result. All comparisons are of signed 32-bit integers.

```
Bxx Vn,Vm,LB ;compare with a variable
Bxx Vn,expr,LB ;compare with a constant
Bxx Vn,[Vm+off],LB ;compare with a table entry
```

xx This is the branch condition. The **xx** stands for: GT=Greater Than, GE=Greater or Equal, EQ=Equal, LE=Less than or Equal, LT=Less Than, NE=Not Equal.

Vn The variable to compare with the next argument.

Vm A variable to compare Vn with or table index variable.

expr A 32-bit integer constant to compare Vn with.

This example collects the latest data value from channel 1 (assumed to be a waveform), waits for it to be in a preset range for 1 second, then outputs a pulse to a digital output bit.

```
START: CHAN V1,1 ; get channel 1 data
      BGT V1,4000,START ; if above upper limit, wait
      BLT V1,0,START ; if too low, wait
IN:    MOVI V2,S(1)/4 ; timeout, 4 instructions/loop
INLOOP: CHAN V1,1 ; to check if still inside
      BGT V1,4000,START ; if above upper limit, wait
      BLT V1,0,START ; if too low, wait
      DBNZ V2,INLOOP ; see if done yet
REWARD: DIGOUT [...1] ; Task done OK
      DELAY S(1) ; leave bit set for 1 second
      DIGOUT [...0] ; clear done bit
      ... ; next task...
```

We want the data to be in range for one second. There are 4 instructions in the loop that tests this, so we set to the loop to run for the number of steps in a second divided by 4. For this to work correctly, the sequencer must be running fast enough so that 4 steps are no longer than the sample interval for the waveform channel.

MOVI

This instruction moves an integer constant into a variable. `MOVIL` is an obsolete instruction that does exactly the same thing. The syntax is:

```
MOVI    Vn,expr,OptLB    ; Vn = expr
```

`Vn` A variable to hold the value of `expr`.

`expr` An expression that is evaluated as a 32-bit integer.

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

`MOVI` is not the same as the `VAR` directive. The `VAR` directive sets the value of a variable when the sequence is copied to the 1401 and does not occupy a step. The `MOVI` instruction is part of the sequence and set the value of the variable each time the instruction is used.

MOV ABS NEG

The `MOV` instruction sets a variable to the value of another with the option of adding a 32-bit number and dividing by a power of two). The `NEG` instruction is identical to `MOV` except that the source variable is negated first. `ABS` is also the same, except that the absolute value (negative values become positive) of the variable is taken first. `ABS` was added at version 7.00. The syntax is:

```
MOV    Va,Vb,expr,shift    ; Va = (Vb + expr) >> shift
NEG    Va,Vb,expr,shift    ; Va = (-Vb + expr) >> shift
ABS    Va,Vb,expr,shift    ; Va = (|Vb| + expr) >> shift
```

`Va` A variable to hold the result. It can be the same as `Vb`.

`Vb` A variable used to calculate the result. It is not changed unless it is the same variable as `Va`.

`expr` An optional expression that is evaluated as a 32-bit integer. If this argument is omitted, it is treated as 0.

`shift` An optional argument in the range 0 to 31, set to 0 if omitted, that sets the number of times to divide the result by 2.

The following examples assume that `V3` holds 1000:

```
VAR    V6,Result
MOV    V1,V3                ; set V1 to 1000
NEG    V1,V3                ; set V1 to -1000
MOV    V1,V3,-8            ; set V1 to 992
NEG    Result,V3,0,4       ; set V6 to -63
ABS    Result,Result       ; set V6 to 63
MOV    Result,V3,4,1       ; set V6 to 502
```

ADDI

This instruction adds a 32-bit integer constant to a variable. `ADDIL` is an obsolete instruction that does exactly the same. There is no `SUBI` as you can add a negative number. The syntax is:

```
ADDI    Vn,expr,OptLB    ; Vn = Vn + expr
```

`Vn` A variable to hold the result of `Vn + expr`.

`expr` An expression that is evaluated as a 32-bit integer.

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs..

The following examples assume that `V1` holds -1000:

```
VAR    V1,Result=-1000
ADDI    Result,1000        ; set V1 to 0
ADDI    V1,-4000          ; set V1 to -5000
```

ADD SUB

The `ADD` instruction adds one variable to another. The `SUB` instruction subtracts one variable from another. In both cases you can optionally add a 32-bit integer constant and optionally divide the result by a power of two. The syntax is:

```
ADD    Va,Vb,expr,shift ; Va = (Va + Vb + expr) >> shift
SUB    Va,Vb,expr,shift ; Va = (Va - Vb + expr) >> shift
```

`Va` A variable to hold the result. It can be the same as `Vb`.

`Vb` A variable to add or subtract.

`expr` An optional expression evaluated as a 32-bit integer. If omitted, 0 is used.

`shift` An optional argument in the range 0 to 31, set to 0 if omitted, that sets the number of times to divide the result by 2.

The following examples assume that `v1` holds -1000, `v3` holds 1000, `v6` holds 100:

```
VAR    V6,Result=100
ADD    V1,V3          ; V1 = 0 (-1000 + 1000 + 0)
SUB    V1,V3          ; V1 = -1000 (0 - 1000 + 0)
ADD    V1,V3,-8       ; V1 = -8 (-1000 + 1000 - 8)
SUB    Result,V3,0,2  ; V6 = -225 (100 - 1000 + 0)/4
ADD    Result,V3,4,1  ; V6 = 389 (-225 + 1000 + 4)/2
```

MUL MULI

`MUL` multiplies a variable by another variable, then optionally adds a 32-bit integer constant and divides the result by a power of two. `MULI` multiplies a variable by a 32-bit integer constant and divides the result by a power of 2.

```
MUL    Va,Vb,expr,shift ; Va = ((Va*Vb)+expr) >> shift
MULI   Va,expr,shift    ; Va = (Va*expr) >> shift
```

`Va` A variable to hold the result. It can be the same as `Vb`.

`Vb` A variable used to calculate the result.

`expr` An expression that is evaluated as a 32-bit integer. It is optional for `MUL` and required for `MULI`. If this argument is omitted, it is treated as 0.

`shift` An optional argument in the range 0 to 31, set to 0 if omitted, that sets the number of times to divide the result by 2.

The following examples assume that `v1` holds -10 and `v3` holds 10:

```
MULI   V1,10          ; V1 = -100 (-10 * 10)
MUL    V1,V3,-8       ; V1 = -992 (-100 * 10 -8)
```

DIV RECIP

`DIV` and `RECIP` divide variables. These are relatively slow instructions; they take around .4 microseconds in a Power1401 mk II, a little less in a Power1401-3, around 1 microsecond in the Power1401 mk I and 3 in a Micro1401.

```
DIV    Va,Vb          ; Va = Va / Vb
RECIP  Va,expr        ; Va = expr / Va
```

If the numerator is 0, the result is 0. If the denominator is 0, the result is 2147483647 if the numerator is greater than 0 and -2147483648 if it is negative. The 1401 truncates all results towards 0. So, $7/3$ or $-7/-3$ is 2, and $-7/3$ or $7/-3$ is -2.

Variable Logic

These instructions perform basic bitwise logical functions between two variables or a variable and a constant while a sequence runs.

AND, ANDI	Bitwise AND of variables, variable and constant
OR, ORI	Bitwise OR of variables, variable and constant
XOR, XORI	Bitwise exclusive OR of variables, variable and constant

AND, ANDI

These instructions bitwise AND a variable with a variable or a 32-bit expression. A bitwise AND means that each bit of the 32-bit result is 1 if both corresponding source bits are 1, otherwise the result bit is 0. For example, 3 AND 1 is 1, 0x55 AND 0xAA is 0.

```
AND    Va,Vb      ; Va = Va AND Vb
ANDI   Va,Vb,expr ; Va = Vb AND expr
```

Va The variable to hold the result.

Vb A variable to AND with *Va* or with the expression.

expr A 32-bit integer constant to AND with *Va*.

This example waits for the digital input to have bit 4 set, then branches based on the digital input value (placed in `VDigIn` or `V56` by `WAIT`).

```
WAIT    [...1....] ; wait for bit 4 set >Wait for Bit 4
ANDI    V1,VDigIn,0x0f ; isolate bits 0..3 (value 0-15)
JUMP    ACTION(V1)    ; branch based on the result
ACTION: JUMP    ACT0   ; action for value 0
        JUMP    ACT1   ; action for value 1
        ...
        JUMP    ACT15  ; action for value 15
```

OR, ORI

These instructions bitwise OR a variable with a variable or a 32-bit expression. A bitwise OR means that each bit of the 32-bit result is 1 if either corresponding source bit is 1, otherwise the result bit is 0. For example, 3 OR 1 is 3, 0x55 OR 0xAA is 0xff.

```
OR     Va,Vb      ; Va = Va OR Vb
ORI    Va,Vb,expr ; Va = Vb OR expr
```

Va The variable to hold the result.

Vb A variable to OR with *Va* or with the expression.

expr A 32-bit integer constant to OR with *Va*.

XOR, XORI

These instructions bitwise exclusive OR a variable with a variable or a 32-bit expression. A bitwise exclusive OR means that each bit of the 32-bit result is 1 if the corresponding source bits differ, otherwise the result bit is 0. For example, 3 XOR 1 is 2, 0x55 XOR 0xAA is 0xff.

```
XOR    Va,Vb      ; Va = Va XOR Vb
XORI   Va,Vb,expr ; Va = Vb XOR expr
```

Va The variable to hold the result.

Vb A variable to XOR with *Va* or with the expression.

`expr` A 32-bit integer constant to XOR with `Va`.

Table access

Tables are declared with the `TABSZ` directive and can be populated with data using the `TABDAT` directive. Most access to tables is through the `[Vn+off]` method, but there are also instructions for loading and storing a variable in a table and for incrementing or decrementing a variable used as a pointer into the table.

`TABLD, TABST` Load a register from the table and store a register to the table
`TABINC` Increment a register and branch while within the table

TABLD TABST

These two instructions load a variable from the table and store a variable into the table. Many instructions can load arguments from the table, so `TABLD` is not often required.

```
TABLD Vm, [Vn+off], OptLB ; load Vm from the table
TABST Vm, [Vn+off], OptLB ; store Vm into the table
```

`Vm` The variable to load from the table or store into the table.

`+off` An optional expression that evaluates to an integer in the range -1000000 to 1000000 . If omitted, 0 is used.

`Vn` The variable value plus the offset is used as a table index. If the index lies in the table, `Vm` is loaded from the table or stored in the table at the index. If the index is not in the table, `TABLD` copies 0 to `Vm` and `TABST` does nothing.

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

TABADD TABSUB

These two instructions add a table value to a variable or subtract a table value from a variable. These instructions were added at version 7.00.

```
TABADD Vm, [Vn+off], OptLB ; add table value to Vm
TABSUB Vm, [Vn+off], OptLB ; subtract value from Vm
```

`Vm` The variable to add data to or subtract it from.

`+off` An optional expression that evaluates to an integer in the range -1000000 to 1000000 . If omitted, 0 is used.

`Vn` The variable value plus the offset is used as a table index. If the index lies in the table, `Vm` is changed. If the index is not in the table, the instruction does nothing.

`OptLB` If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

TABINC

This instruction adds a constant to a variable and detects if the result is a valid table index. If it is a valid index, the instruction branches. If it is not, the result is reduced by the table size if it is positive and is increased by the table size if it is negative, and the instruction does not branch. This gives you an efficient way to work through the table.

```
TABINC Vn, expr, OptLB
```

`Vn` This variable is assumed to hold a valid table index.

`expr` This expression evaluates to a positive or negative number that is added to `Vn`.

`OptLB` If present, branch to this label if `Vn+expr` is a valid table index.

For example, the following codes plays pulses through DAC 0 based on data in the table. The table data holds

groups of three items, holding the time for the DAC to stay at 0, the DAC amplitude and the time to stay at the amplitude. Some example table data is given, but the data could also be set with the `SampleSeqTable()` script command.

```

      SET      0.100 1 0      ;run at 10 kHz
      TABSZ   12              ;4 sets of 3 items
      TABDAT  ms(50)-2,VDAC32(1),ms(50)-3
      TABDAT  ms(100)-2,VDAC32(1.3),ms(70)-3
      TABDAT  ms(200)-2,VDAC32(1.5),ms(90)-3
      TABDAT  ms(400)-2,VDAC32(1.9),ms(110)-3
LOOP:  'G MOVI   V1,0          ;use V1 as the table pointer
      DAC      0,0            ;strt with the DAC low
      DELAY    [V1]           ;wait for first period>Low
      DAC      0,[V1+1]       ;get the DAC value
      DELAY    [V1+2]         ;wait for second period>High
      TABINC   V1,3,LOOP
      DAC      0,0            ;tidy up the dac

```

Access to data capture

Most activities in the sequencer are independent of the sampling process. However, there are times when you need to know the value of a channel to decide what to do next. The `CHAN` command gives you the latest waveform value or number of events on a channel. The `TICKS` command tells you the current time in terms of the sampling clock ticks.

The sequencer can also send information in the other direction. If the digital marker channel is active you can force it to record the current digital input state with `REPORT`, or you can force it to record a marker of your own choosing with `MARK`.

```

REPORT,MARK      Simulate an external E1 pulse to record/set a digital marker
CHAN              Get the latest waveform value or event count from a channel
TICKS            Load a variable with the time in Spike2 time units
TICK0            Sets the zero time used as a reference for TICKS, new at 8.00

```

CHAN

This instruction gives the output sequencer access to sampled data on a waveform channel and to the number of recorded events on all other channel types. You can also use this command to get the most recent value written to the DAC outputs. The variable value is 0 if the channel is not being sampled. You cannot use this command on channels 30 and 31 (Text marker and Keyboard marker) as these channels do not exist in the 1401.

```

      CHAN    Vn,chn          ; Vn = ChanData(chn)

```

`chn` The channel number is 1 to 100 for sampled channels or 0 to -7 for the last value written by the sequence to DAC 0 to 7. The result is the most recent data available. For a slow waveform channel this could be a long time in the past. In triggered sampling mode, waveform data is available between triggers.

Waveform and DAC data are treated as 16-bit signed values from -32768 to 32767. You also have more efficient access to DAC values as 32-bit data in variables `V57` to `V64` (`VDAC0` to `VDAC7`) without the need to use the `CHAN` instruction.

This example waits for a signal to cross 0.05 volts and produces a pulse. We assume that channel 1 is a waveform.

```

      SET      0.100 1 0      ;run at 10 kHz
      VAR      V1,level=VDAC16(0.05) ;level to cross
      VAR      V2,data        ;to hold the last data
      VAR      V3,low=0       ;some sort of hysteresis level
      DIGOUT   [00000000]     ;set all dig outs low
BELOW:  CHAN   data,1         ;read latest data   >wait below
      BGT     data,low,below ;wait for below   >wait below
ABOVE:  CHAN   data,1         ;read latest data   >wait above
      BLE     data,level,above ;wait for above   >wait above
      DIGOUT  [...1]         ;pulse output...
      DIGOUT  [...0],below ;...wait for below

```

DAC values and arbitrary waveform output

You can only read back the current DAC value when you set it with a sequence instruction (DAC, ADDAC, RAMP, Cosine output). If you use arbitrary waveform output, this uses a separate, more efficient mechanism; the price you pay for the increased efficiency is less knowledge of exactly what the DACs are doing.

Event count overflow in version 8

You should be aware that when using CHAN to count events it is possible for the event count to exceed the 32-bit range of the variable. That is, the count will start at 0, then increase up to 2147483647 (0x7fffffff, the most positive number in 32-bit twos complement coding). The next event increments the number to 0x80000000, which is -2147483648 (the most negative number in 32-bit twos complement coding). The value will continue to increment by getting more positive and will reach 0 (again) at 4294967296 events, after which the cycle will repeat.

For example, if your events have an average rate of 100 Hz, the first wrap around happens after some 500 hours (more than 20 days). Remember that if this is likely to be a problem for you, you can probably reorganise your code to use differences of event counts.

TICKS

This instruction sets a variable to the current sampling time past the current zero time set by the TICK0 instruction in Spike2 time units (microseconds per time unit set in the sampling configuration) and adds an expression or 0 if `expr` is omitted. It can detect if the result does not fit in a signed 32-bit variable. Typical use of this instruction is to wait for a defined time while performing other sequencer actions, avoiding the need to carefully count sequencer instructions.

	TICKS	<code>Vn,expr,ovLab</code>	<code>; Vn = Spike2 time + expr, branch on overflow</code>
<code>Vn</code>		A variable to hold the time past the current zero time plus <code>expr</code> . This may NOT be <code>V255</code> or <code>V256</code> as these are used to save the current zero time. Remember that variables are 32-bit signed numbers, but the time in clock ticks can be up to 64-bits in size. This means that there is a limit to the result that can be stored in <code>Vn</code> . Put another way, the result in <code>Vn</code> can overflow (see <code>ovLab</code> for a way to detect this). If overflow occurs, the value in <code>Vn</code> is full scale maximum (2147483647 or 0x7fffffff) if the result is too large to fit in 32 bits or full scale minimum (-2147483648 or 0x80000000) if the result is too negative.	
<code>expr</code>		An expression that evaluates to a constant 32-bit signed number of clock ticks that are to be added to the result of the current time less the current zero time set by TICK0. This expression may be omitted in which case the value 0 is added. The <code>sTick()</code> , <code>msTick()</code> and <code>usTick()</code> expression functions can be used to make the sequence independent of the microseconds per time unit value.	
<code>ovLab</code>		If present, this is a label that is branched to if the result of "current time - zero time + <code>expr</code> " does not fit in a 32-bit signed variable. If omitted, the next instruction runs regardless of overflow.	

This can be used with the CHAN command and variable related branches to check the timing of external pulses. The sequencer runs under interrupt, and competes for time with other interrupt driven processes in the 1401 interface. This causes some "jitter" in the timing. The jitter is typically only a few microseconds.

Changes at Spike2 8.00

Because the time in Spike2 clock ticks no longer fits in a 32-bit variable, you should be aware that the result can overflow. To allow for this, the command is extended as described above to allow you to detect overflow and the new TICK0 sequencer command and the `SampleSeqTick0()` script command are added to give you control of where the TICKS result is relative to.

TICK0

This instruction sets the zero time used as a reference for the TICKS instruction to the current sampling time in Spike2 clock ticks. This is new in Spike2 version 8 and is required because sampling can now run for many more clock ticks than can be stored in a 32-bit sequencer variable.

If you have existing sequences, and you sample for less than 2147483648 clock ticks you can ignore this command and the old sequences will continue to work. However, if you sample for longer than this (31 minutes at 1 microsecond resolution, 5 hours at 10 microseconds resolution...) and you use `TICKS`, you should rewrite your old sequences to periodically use `TICK0` and time relative to this.

The zero time is stored in `V255` (bits 0-31) and `V256` (bits 32-63). Be very careful if you modify these variables using other sequencer instructions if you want `TICKS` to give useful results.

`TICK0` *expr*, *OptLB*

expr An expression that evaluates to a constant 32-bit signed number of clock ticks that are to be added to the current time to generate the new zero time. This expression may be omitted in which case the value added is 0. The `sTick()`, `msTick()` and `usTick()` expression functions can be used to make the sequence independent of the microseconds per time unit value.

OptLB An optional label that sets the next instruction to run.

REPORT MARK

The `REPORT` instruction records a digital marker (if the digital marker channel is enabled) as if there were an external pulse that triggered a digital marker. The `MARK` instruction does the same, except it takes the argument as the value to record. `REPORT` has no arguments.

`REPORT` *OptLB*
`MARK` *expr* | *Vn* | [*Vn+off*], *OptLB*

expr The argument should have a value in the range 0 to 255. If a variable or table is used, the bottom 8 bits of the value are used.

OptLB If this label is present it sets the next instruction, otherwise the next sequential instruction runs.

```

WAIT    [.....1]    >Waiting for bit 0
REPORT                                     ;save a marker when this is set
MARK    12          ;set code 12 as a digital marker

```

Time resolution

There is a restriction in Spike2 that events on a channel should all be at different times. If the Spike2 data file is running with a time resolution that is similar to the speed of the output sequencer, it is possible that two consecutive `MARK` or `REPORT` instructions could be logged at the same time. This will cause Spike2 to report an error when sampling ends. If you get such an error you must either reduce the data file time resolution or increase the gap between the sequencer instructions.

Randomisation

These functions use a pseudo-random number generator. The generator is seeded by a number that is based on the length of time that the 1401 has been switched on.

`BRAND`, (`BRANDV`) Random branch with a probability
`MOVRND` Load a variable with a random number
`(LD1RAN)` Load counter 1 (`V61`) with a random number 1-256

BRAND

`BRAND` branches with a probability set by the argument or by a variable. This could be used when several different stimuli are required, but in a random sequence. `BRANDV` is an obsolete name for the same instruction.

`BRAND` *LB*, *expr* | *Vn* | [*Vn+off*]

LB Where to go if the branch is taken

expr This is the probability of branching in the range 0 up to (but not including) 1.

Vn When a variable or table entry is used for a branch, the value is treated as a 32-bit unsigned number; 0 means a probability of zero and 4294967295 (the largest 32-bit unsigned number) means

a probability of 0.9999999998.

```
BRAND LB,0.5 ;branch with 50% probability
```

To produce a multiple way random branch you use more BRAND instructions. A three way equal probability branch to LA, LB and LC can be coded:

```
BRAND LA,0.33333 ;Split the first route with p=1/3
BRAND LB,0.5 ;0.6667 to here * 0.5 is 0.3334 (1/3)
LC: ... ;If neither of the above, comes here
```

The following shows the sequence for a five-way branch with equal probabilities:

```
BRAND LA,0.2 ;5 way, LA probability is 0.2 (1/5)
BRAND FX,0.5 ;Probability to here=0.8, so to FX=0.4
BRAND LB,0.5 ;Probability to here=0.4, so to LB=0.2
LC: ... ;Probability to here=0.2
FX: BRAND LD,0.5 ;Probability to here=0.4, so to LD=0.2
LE: ... ;Probability to here=0.2
```

The best technique is to reduce the branches to a power of two as soon as possible. Case 1 of the five-way branch is split off (probability of 0.2), leaving 4 ways. The 4 ways are split with a probability of 0.5 (0.4 for each division) then the last two routes are split, again with a probability of 0.5 (0.2 for each division).

Poisson process

In a Poisson process, the probability of something happening per time interval is constant. You can generate a delay with a Poisson statistic by:

```
POISSON: BRAND POISSON,prob ; poisson delay
```

The probability is given by $prob = 1.0 - 1.0/(mDelay * S(1))$, where `mDelay` is the mean delay required in seconds and `S(1)` is the built in function that tells us how many steps there are per second. If you would rather express this in terms of a rate, then $prob = 1.0 - rate/S(1)$, where `rate` is the expected rate in Hz.

```
TENHZ: BRAND TENHZ,1.0-10/S(1) ;10 Hz mean rate
DIGOUT [.....1] ;set output high
DIGOUT [.....0],TENHZ ;set output low, goto TENHZ
```

This example generates a digital output that pulses to produce an approximation to a Poisson distributed pulse train with a mean frequency of 10 Hz. The approximation improves the shorter the step time. The mean interval between pulses is 100 milliseconds plus the time for 2 steps and the shortest gap between pulses is 3 sequencer steps.

Scripts and variables

From a script you can set sequencer variables as 32-bit signed integers. For the range 2147483648 to 4294967295 we must use negative numbers. This script example shows you how to convert a probability into a variable value and pass it to the sequencer:

```
Proc SetBrandVar(prob, v%) 'prob is probability, v% is variable
prob *= 4294967296.0; 'range 0-4294967296 is 0 to 1.0
if (prob > 4294967295.0 ) then prob := 4294967295.0 endif;
if prob > 2147483647 then prob -= 4294967296.0 endif;
if prob < -2147483647 then prob := -2147483647 endif;
SampleSeqVar(v%, prob);
end;
```

MOVRND

This instruction generates a random number in the range 0 to a power of 2 minus 1, then adds an integer constant to it and stores the result in a variable.

```
MOVRND Vn,bits,expr
```

`Vn` The variable to hold the result.

`bits` The number of random bits to generate in the range 1 to 32. The generated random bits fill the variable starting at the least significant bit. Bits above the highest numbered generated bit are set to 0.

`expr` An optional expression that evaluates to a 32-bit integer number, that is added to the random bits. If this is omitted, nothing is added.

Expressed in terms of the script language, the random number is one of the numbers in the range `expr` to `expr + Pow(2, bits) - 1`. For example, `MOVRND V61, 8, 1` emulates the obsolete `LD1RAN` instruction that generates a random number in the range 1 to 256.

The following code fragment implements a random delay of between 1 and 2.024 seconds (assuming a 1 millisecond clock).

```
MOVRND V1,10,998 ;load V1 0 with (0 to 1023) + 998
DELAY V1 ;this uses 999 to 2023 steps
```

LD1RAN

This obsolete instruction loads counter 1 (`V61`) with a pseudo-random number in the range 1 to 256. It is implemented as `MOVRND V61, 8, 1`. Do not use this instruction in modern code. `V61` is also used to store the last value written to DAC 4.

Arbitrary waveform output

In addition to generating voltage pulses, ramps and cosine waves through the DACs, Spike2 can play arbitrary waveforms. The sequencer can start waveform output, test it and branch on the result, stop output or set one more output cycle. Each waveform has an associated code (often a keyboard character). The sequencer uses the code to identify the arbitrary wave to replay. The waveforms are stored in 1401 memory and can be updated during sampling with the `PlayWaveCopy()` script command.

<code>WAVEGO</code>	Start or prepare arbitrary waveform output area
<code>WAVEBR</code>	Test arbitrary waveform output and branch on the result
<code>WAVEST</code>	Start or stop arbitrary waveform output

WAVEGO

The `WAVEGO` instruction starts output from a play wave area, or prepares the output for an external trigger. Starting output can take more time than we want to allocate to a sequencer step so `WAVEGO` sets a flag and output starts as soon as the 1401 has free time (usually within a millisecond). See `WAVEST` for a precisely timed start to output.

```
WAVEGO code{, flags}
```

`code` This is either a single character standing for itself, or a two digit hexadecimal code. This is the code of the area to be played.

`flags` These are optional single character flags. The flags are not case sensitive. Use `T` for triggered waveform output (either an external hardware trigger or using `WAVEST T`). Use `w` to make the sequencer wait at this step until the 1401 has prepared the hardware to play. For example:

```
WAVEGO X ;area X, no wait, no trigger
WAVEGO 23,T ;area coded hexadecimal 23, triggered
WAVEGO 0,WT ;area 0, wait until trigger armed
WAVEGO 1,W ;area 1, wait until play started
```

If you need to know when the output started, use the `WAVEBR T` option. If you need to know that the request to start the playing operation has been honoured, but you do not want to hang up the sequencer with the `w` option, use the `WAVEBR W` option.

`WAVEGO` cancels existing output just before the new area starts to play. If you use `WAVEBR` after the play request, unless you use the `WAVEGO` or `WAVEBR W` option to be certain that the new area is active, the `WAVEBR` result may be based on the previous area.

WAVEBR

The WAVEBR instruction tests the state of the waveform output and branches on the result. No branch occurs if there is no output running or requested.

```
WAVEBR LB,flag
```

LB Label to branch to if the condition set by the flag is not met.

flag An optional single character flag to specify the branch condition:

W branch until a WAVEGO request without the w flag is complete.

C branch until the play wave area or cycle count changes.

A branch until the play wave area changes.

S branch until the current output stops.

T branch until output started with WAVEGO begins to play.

The following sequence tracks the output when we have two play areas labelled 0 and 1. Area 0 is set to play 10 times and is linked to area 1. The sequence below will track the changes. Play wave DAC output happens one play wave clock tick after the output is changed, so the sequencer can know that a DAC output is about to change.

```

WAVEGO 0,WT ; area 0, wait for armed
MOVI V1,5 ; load variable V1 with 5
WT: WAVEBR WT,T ; wait for external trigger>Trigger?
W5: WAVEBR W5,C ; wait for cycle >Waiting for cycle
DBNZ V1,W5 ; do this 5 times >Waiting for cycle
WA: WAVEBR WA,A ; wait for area >Wait for area
WE: WAVEBR WE,S ; wait for end >Wait for end

```

The WAVEGO command requests a triggered start and waits until the trigger is armed before moving on. It then waits for an external trigger at the WT label. Next the sequence tracks the end of 5 output cycles. At label WA the sequence waits for the area to change and finally the sequence waits for output to stop.

WAVEST

The WAVEST instruction can start output that is waiting for a trigger and stop output that is playing, either instantly, or after the current cycle ends.

```
WAVEST flag
```

flag This are optional single character flag and specifies the action to take:

T Trigger a waveform that is waiting for a triggered start.

S Stop output immediately, no link to the next area.

C Play to the end of the current cycle, then end this area. If there is another area linked it will play.

The following code starts output with an internal trigger and then stops it

```

WAVEGO X,TW ; arm area X for trigger
WAVEST T ; trigger the area
DELAY 1000 ; wait
WAVEST S ; stop output now

```

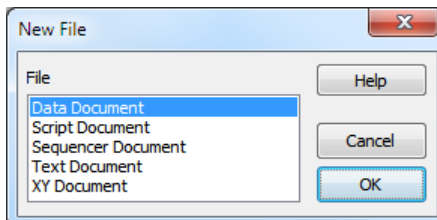
6: File menu

File menu

The File menu is used for operations that are mainly associated with files (opening, closing and creating) and with printing.

New File

This command creates a new Spike2 data file for sampling based on the current sampling configuration, an output sequence file, a script file, text file or an XY file. The command opens a new file dialog box in which you choose the type of file to create. You can activate this command from the menu and from the toolbar. The script language equivalent is `FileNew()`.



You can make files of five types: data, output sequencer (pulse) files, script files, text files and XY files. Double-click a file type, or select a file type and click OK, Spike2 will open a window of the specified type. Each file type has its own file name extension.

Data Document

A sampling window opens plus additional windows set by the sampling configuration. Data documents are not stored in memory, unlike most new files, but are kept on disk. Until they are saved after sampling they are temporary files in the directory set in the Edit menu Preferences. The file name extension is `.smrx` (or `.smr` for the legacy format). We are often asked why the file extension is `smr` and not, for example, `spk`. It stands for son of MRate - MRate was a DOS program that was the predecessor of Spike2 in that it sampled both waveforms and event times. Only one sampling document can be open at a time.

Sequencer Document

A new window opens in which you can type, edit and compile an output sequence. The file name extension is `.pls`.

Text Document

Text files can be used to take notes, build reports and to cut and paste text between other windows and applications. The file name extension is `.txt`.

Script Document

A script editor window opens in which a new script can be written, run and debugged. The file name extension is `.s2s`.

XY Document

XY windows are used to draw user-defined graphs with a wide variety of line and point styles. The Windows file name extension is `.sxy`. They can be generated by scripts, or from the Analysis menu for trend plots.

Other types used by Spike2

In addition to the document types listed above, Spike2 also uses:

Result view files

These hold result views (file extension `.srf`). Result views are created as the result of an analysis, or from the script language, not with the File menu New command. Version 8 reads all result view files written by previous versions of Spike2. We write result view files in the oldest format compatible with the data for backwards compatibility. Files holding raster data with event times that exceed the 32-bit timing capability of older versions of Spike2 are only readable by Spike2 version 8.

Resource files

Spike2 version 8 creates and reads resource files with the extension `.s2rx`. It will also read old-format resource files with the extension `.s2r`. Resource files are associated with data files of the same name, but with the extension `.smr`, `.srf` and `.sxy`. They hold configuration information so that Spike2 can restore the display. These files are not essential to Spike2 and if deleted, the associated data file is not damaged in any way.

Configuration files

Spike2 stores sampling configuration information in files with the file name extension is `.s2cx`. Spike2 will also read old format files with the extension `.s2c`.

Filterbank files

These files hold descriptions of digital filters and have the extension `.cfbx`. We also read the older format `.cfb` files if we cannot find a modern file. They are used by the Analysis menu Digital filters command.

Multimedia files

These files have the extension `.avi`. They are created by the separate `s2video` program and are opened by the View menu Multimedia Files command.

Compatibility with Spike2 version 7

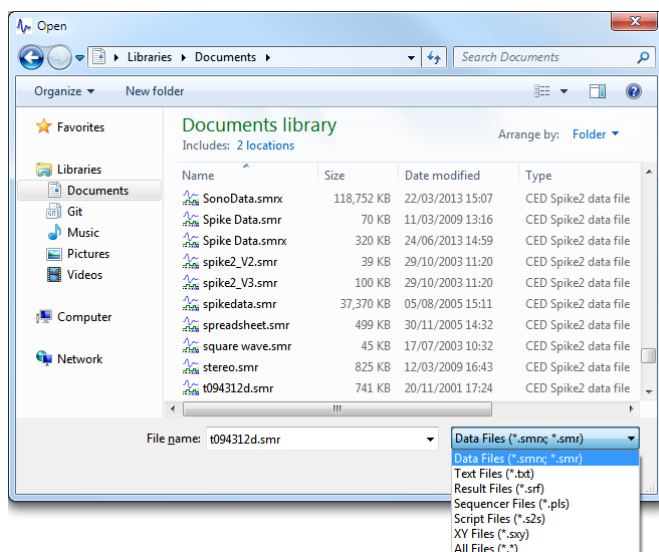
The `.s2rx`, `.s2cx` and `.cfbx` files hold their contents in XML format. Spike2 version 7.11 onwards can read these files; earlier versions of Spike2 cannot. The compatibility of `.srf` files is described above.

Open

This file menu and toolbar option opens a dialog in which you can select a file. The dialog features depend on the operating system. You can open six file types with this command: a Spike2 data file, a Spike2 result view, a text file, a script file, an output sequence file or an XY file. The type of the file is selected with the Files of type field.

When you select a data file, Spike2 looks for a resource file of the same name, but with the file extension `.s2rx` or `.s2r`. If this is found, the new window displays the file in the same state and screen position as it was put away. Several windows may open if the file was closed with the `Ctrl` key held down. See the Close command details.

Spike2 open an appropriate window type to match the selected file type.



Read only files

You can open Spike2 data files that are read-only, but you are not allowed to write any changes back to the file. Read-only data files have [Read only] added to the window title. Unless the file is on a read-only medium (such as CD) you can usually clear the read-only state by opening the Property page of the file (right click on the file name or a list of selected files) and clear the Read-only check box. If you copy files from a read-only drive such as a CD, the copied file is marked read-only.

Read-only text-based files open normally, but you cannot edit them.

Import

Spike2 can translate data files from other formats into Spike2 data files. The File menu Import command leads to a standard file open dialog in which you select the file type and file name to convert. You then set the file name for the result; Spike2 will suggest the same name with the extension changed to `.smrx`. The details of the conversion depend on the file type.

Supported formats include text files with data in columns, the CFS files used by CED programs such as Signal and SIGAVG, Spike2 for Macintosh files as well as data from many third party vendors. Spike2 searches the `import` folder in the Spike2 installation folder for CED File Converter DLLs. If you need to translate a file format that is not covered, please contact us and describe your requirements. The script language command to convert files is `FileConvert$()`.

We will write the new files as a 64-bit `.smrx` file. If you want the output as a 32-bit `.smr` file you can export the result, or use Spike2 version 7 to import the data.

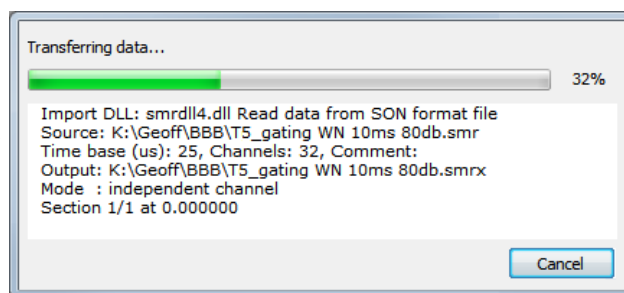
Data import has two main phases:

1. Scanning the input files to detect the file contents, channel types and data and time ranges present.
2. Transfer of data into a Spike2 data file

The scanning phase is handled by the file converter DLL. Data transfer is a co-operative effort between Spike2 and the converter DLL.

Display during import

Data file import can take a considerable time, especially if the source is very large or spread over many files. Spike2 opens a resizable progress dialog displaying a summary of the import procedure; it also gives you the option of cancelling the import. This dialog may be unresponsive with older importers, particularly during the scanning phase. Modern importers give feedback during scanning but in some cases, you may not be able to cancel import until the data transfer phase begins.



The progress dialog closes when the file has been transferred; the text contents of the dialog are written to the Log view when the file closes, which can be useful if there were problems during the import.

Importer Configuration Files

Some importers store configuration files in the import folder; these files have the extension `.icf`. The last used configuration for importer `XXX` is saved in `XXX_Last.icf`. If a default format exists, it is saved in `XXX_Def.icf`. If the `_Def.icf` file exists, it is used, otherwise the `_Last.icf` file is used. Most importers will survive without any configuration information as the built-in default settings will do something useful. However, if you intend to import a binary or text file with the `FileConvert$()` script command it is essential that you have created configuration files, otherwise these importers will not know what to do with the data.

64-bit Spike2 build problems

Some of these importers rely on DLLs or object libraries supplied by third parties. If these DLLs or libraries only exist in 32-bit versions, the 64-bit build of Spike2 cannot use them. These are marked as "32-bit only" in the Notes (below); to import data for these you must install the 32-bit version of Spike2.

DLL name

The DLL column in the table below holds the name of the import DLL in the import folder and is provided for script users who wish to force a particular DLL to be used to import your file.

File types we import

We import data from a wide variety of file types and we attempt to preserve as much of the original data content and accuracy as we can. If there is a file type that you would like imported that is not in the list, please

contact us. To import a new type we need accurate file format information. We would like to thank the companies that made such information available to us for their help. Some companies do not release their file formats; in a few cases we have managed to figure out enough to extract useful information. In cases where companies do not release the information and we have not been able to figure it out, you can sometimes export data in other formats that we can import; of course, every change of format will tend to reduce the quality and quantity of data that is imported.

If you find a problem in any importer, please contact us and explain the problem; in many cases the importers were written with only a small number (sometimes 1) of examples of the source data. We will need a source data file that fails to convert.

Data source	Extensions	DLL	Data types imported	cmd\$	Notes
Alpha MED Sciences	DAT,MODAT	med64	Waveform		Conductor, Performer and Mobius data. Can import files larger than 2 GB.
Alpha Omega Engineering	MAP, MPX	alpha	Waveform, WaveMark, RealMark, Event		Old format event data and WaveMark supported. TextMark, MPX supported. Fixed a y axis scaling problem in Jan/2014.
Axon Instruments (Molecular Devices)	DAT,ABF	axon	Waveform, TextMark		32-bit only. Imports 16-bit integer and 32-bit float. Library updated Jan/2014.
Axona	BIN	axona	Waveform		
Binary data	BIN,DAT	binary	Waveform, RealWave	Yes	Can import 8/16/32/64 bit signed and unsigned integers data in big and little-endian formats.
Bionic/Cyberkinetics	NEV,RND,NS*bionic		Waveform, WaveMark, Event		Can import files larger than 2 GB.
Biopac	ACQ	biopac	Waveform, TextMark		32-bit only. Reads version 45 files and older. Imports 16-bit integer and 32-bit float data. Library updated Jan/2014.
CED CFS	DAT,CFS	cfsdll	Waveform, RealWave		Jan/2014: corrected marker channel times when block has a negative time offset.
CED SON (Macintosh)	SMR,SON	sondll	Waveform, WaveMark, TextMark, Event		
COLD_Datein	*	colddll	Waveform, TextMark		Pulsion Medical Systems.
CONSAM	SSD,DAT	consdll	Waveform		From Prof. D. Colquhuon. Supports version 1001, 1002.
DATAQ Instruments	ACQ	codas	Waveform, TextMark		(Codas)
DataWave	UFF,CUT	EWBufdll	Waveform, WaveMark, TextMark, Event		Imports both Discovery and Workbench files.
DATAPAC	PAR,PBR,PCRdatapaq		Waveform, RealWave, Event		
Data Sciences International	*	dataq dataqv5	Waveform, RealMark, TextMark		32-bit only. You need a dongle to import this file format. Contact CED for details. Files with 4 character extensions supported. Can import all

Data source	Extensions	DLL	Data types imported	cmd\$ Notes
				files up to version 5. You should use the dataqv5 importer unless you have problems.
Delsys Files	EMG	delsys	Waveform	Version 4 supported.
elmiko medical	*	elmiko	Waveform	
European Data Format(+)	EDF, BDF	edf	Waveform, TextMark	Since Spike2 version 7.06 we import dates so that time of day display can be used. Imports 16 and 24-bit data.
Grass-Telefactor	BIN	polyview	Waveform, TextMark	(PolyView) We import versions from 2.0
HLR Data Format	HLR	hlr	Waveform, RealMark, Event	From software Platon and Pyton version 3.00 and 4.00.
Heka Data Format	DAT	heka	Waveform	Since Spike version 7.06 we support old format Apple Macintosh files. 7.06 also supports time of day display mode.
Intan	*	intandll	Waveform	
Multi Channel System (Mc_Rack)	MCD	mcddata	Waveform, WaveMark, Event	32-bit only. We can import 12 and 16-bit data. Library updated Jan/2014.
MindSet (16/24) data Files	BIN	mindset	Waveform	MindSet, MindMeld
MindWare data files	MW	mindw	Waveform	
Motion Lab Systems	C3D	c3d	Waveform	
Native ADC	ADC	nadcdll	Waveform, TextMark	Used by Polish universities as a common data format.
NeuroScan	EEG, CNT	neuro	Waveform, Event	Can read both 16 and 32-bit data.
NewBehavior	HEX	neurolog	Waveform	(Neurologger) old and 2012 formats. Jan/2104: added a dialog to select the recording session to import.
Neuralynx	NEV, NCS, NTTnlynx, NSE, NST		Waveform, WaveMark, TextMark	Waveform channel y scaling factor corrected Jan/2014.
Plexon	NEX, PLX, DDTplexon		Waveform, WaveMark, TextMark, Event	Can read data generated by the Plexon version 1.07 library.
RC Electronics	PRM, INX, DATrcelectr		Waveform, TextMark	Imports 12 and 16-bit data.
Text files	TXT, ASC	asciidll	Waveform, RealWave, Event, TextMark	Fixed a TextMark timing issue in Jan/2014.
TMS International	S00	tmsi	Waveform, Marker	Marker format is untested as we have no example of it.
Tucker-Davis	TSQ	tdtdll	WaveForm, WaveMark,	You also need the .tev

Data source	Extensions	DLL	Data types imported	cmd\$ Notes
Technologies			Event, Marker, RealWave	file to import the data.
WAV (Microsoft)	WAV	mwave	Waveform	
WaveMetrics Igor Pro	IGO, IBW, PXP	Igor	Waveform	Imports both PC and Mac files up to and including version 5.

Binary importer commands

Uses configuration files `Bin_Last.icf` and `Bin_Def.icf`. If the `.icf` files are not found, old format `.cim` files are used.

The binary importer supports the following names in the `cmd$` string for the `FileConvert$()` command.

name	value
<code>Conf</code>	Configuration file name. If this string is not empty then the provided file name including its full path will be used to load the configuration file. This keyword is always applied first, regardless of its position in the <code>cmd\$</code> string.
<code>Bigend</code>	Input file origin. Set to 0 if this is a little-endian file (which is almost always the case on modern systems) or to 1 if this is a big-endian file. The default is 0. In a little-endian file, lower significance bytes are at lower file offsets than higher significance bytes. If the 32-bit hexadecimal number <code>0x87654321</code> was stored at the start of a binary file, the first 4 bytes would be <code>0x21</code> , <code>0x43</code> , <code>0x65</code> , <code>0x87</code> in a little-endian file and as <code>0x87</code> , <code>0x65</code> , <code>0x43</code> , <code>0x21</code> in a big-endian file.
<code>Type</code>	Input data type. Set to 0 for 8-bit signed integers, 1 for 8-bit unsigned integers, 2 for 16-bit signed integers, 3 for 16-bit unsigned integers, 4 for 32-bit signed integers, 5 for single-precision floating point numbers and 6 for double-precision floating point numbers. The default is 2.
<code>Rate</code>	File sampling rate in Hz. If present it sets the overall file sampling rate. The default rate is 1000 Hz.
<code>Chans</code>	Sets the number of imported waveform channels. The default is 3.
<code>Names</code>	Channel name. You provide a list of channel names for each imported channel separated by commas, for example: <code>Names=Resp,HR,BP</code> . The default name is <code>Chan(n)</code> where <code>n</code> is the channel number.
<code>Units</code>	Channel units. You provide a list of units for each imported channel separated by commas, for example <code>Units=bpm,bpm,mmHg</code> . The default units are <code>v</code> .

Resources are used in the following order:

1. A default state is set.
2. `Bin_Def.icfx` is used, if it exists. If not found, `BinDef.icf` is used, if it exists.
3. If no `Bin_Def` files are found, `Bin_Last.icfx` is loaded, and failing that, `Bin_Last.icf`
4. If the `Conf` keyword is used, and the nominated file exists, it is loaded and overrides the current setup.
5. Any other keywords are then applied.

An example

```
var ret$;
var src$, dest$, flag%, err%, scom$;
src$ := ""; 'Blank name so user is prompted for the source file
dest$ := ""; 'Blank name so user is prompted for the destination file
flag% := 0;
scom$ := "Conf=c:/s/b.icfx;Bigend=0;Type=3;Rate=500;Chans=1;Names=HR,BP;Units=bpm,mmHg";
ret$ := FileConvert$(src$, dest$, flag%, err%, scom$);
```

Import DLL versions

Spike2 version 8 supports two different import DLL versions: version 4 and version 5. Version 4 DLLs are the same as those used by Spike2 version 7 and are limited to importing 32-bit time ranges and the output file size is limited by the 32-bit SON file size limit of 1 TB. The version 5 DLLs are newly designed for Spike2 version 8 and support 64-bit times and file sizes limited only by your disk sizes and patience (though some importers may be limited by available system memory). The version 5 import DLLs can also provide much better feedback into the progress dialog during the scanning phase (though this is up to the DLL author).

When you select a file to import, you must first select a file type. If the file convert DLL is a version 4 converter, we add [4] to the end of the file importer. For example, if you had a version 4 importer for old Spike2 data files the file type list would display:

Spike2 files (*.SMR)[4]

To indicate that this was an old style importer. We intend to convert most importers to the new format, but this will take us some time to accomplish.

Writing an import DLL

If a data format is generally available and sufficient users want to import it, we are usually prepared to write an import DLL to support the format. However, you may have an exotic format of your own than no-one else uses, or you may have a private format that you do not wish to reveal to third parties. In these cases, you may wish to write your own import DLL. To do so you will need to be able to write code in C or C++ in the Windows environment. We have documentation available on the DLL interface. Contact CED for more information.

Import problems

If there are disk read problems or the input is damaged the import is likely to end with an error message. However, there are other problems that can occur during data import that we can work around.

Data is input section by section. Some files may have only one section, others may have many. A section can refer to a single sweep of data from a sweep-based capture system or it may refer to data captured after a particular time marker; it is a flexible concept.

Within each section, data is captured channel by channel in ascending time order.

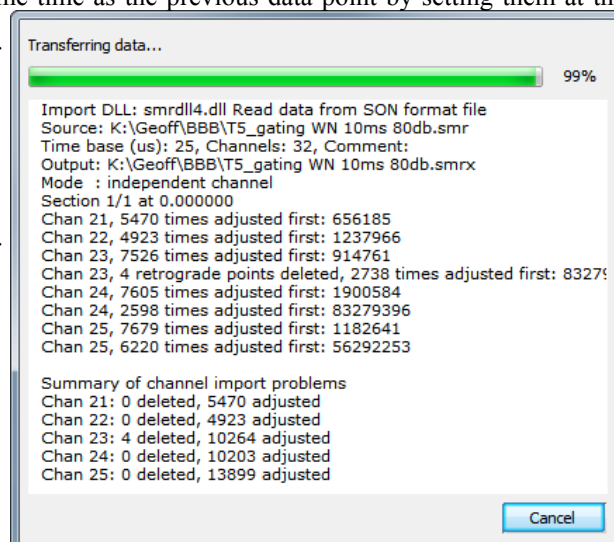
The output data is written in strictly ascending time order into the Spike2 data file. Spike2 data files have the rule that within a channel, all data is in ascending time order and no two items can occur at the same time. There is a basic quantum of time, the underlying clock tick, that all items are timed with. This is typically of order 1 microsecond and usually no more than 50 microseconds; it can be much smaller.

When we import data, we check that all the data meets these requirements. When we import event-based channels (event, level event, marker, WaveMark, TextMark and RealMark data), we read the data block by block. We perform the following operations on each block:

1. If we have to apply a time offset, we do so (this can happen when importing sweep-based data where each sweep has an independent time base)
2. We check that the data is in ascending order of time
3. If not in ascending time order we sort it into order
4. We delete data at negative times or at times that are beyond the tick range supported by Spike2
5. We delete data that occurs before the data at the end of the previous data block on this channel

6. We adjust the times of data that occurs at the same time as the previous data point by setting them at the time of the previous data plus one tick. This type of adjustment usually occurs as the result of "bouncy switches", but this can occur legitimately. For example, if a system wants to drop a lot of text at a particular time it might code the text in 80 character chunks all with the same time stamp.

This is an example of a data file with two types of problem. As we process each data block we generate a message warning about deleted items and adjusted times. Where a time is adjusted, we display the time (in clock ticks) of the first adjusted point in the block. This can help you locate it afterwards to diagnose the problem. We will display up to 100 warnings per channel, after which the file is processed silently. If there are any adjustments needed, a Summary section is added to the output (and to the Log file).



Text importer

When a user imports a text file interactively, the last user configuration is saved as an XML format file called `Ascii_Last.icfx` in the current users application data path.

At Spike2 7.10, the Time column units were corrected and we improved channel on/off, type and sample rate management.

Global Resources

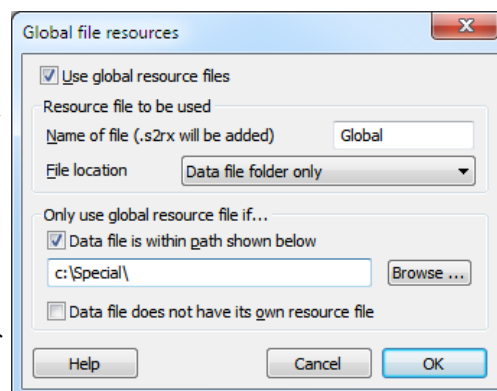
If the current view is not a time, result or XY view, this command appears in the menu, otherwise it appears in the Resource Files popup menu.

Normally, each Spike2 time, result and XY data file has an associated resource file with the same name and `.s2rx` or `.s2r` extension. These per-file resources remember the screen layout, cursor positions, active cursor parameters and other settings. They allow each file to open with exactly the same screen appearance it had when it closed at the cost of an extra file on disk.

However, per-file resources do not let you use the same display and cursor settings for a sequence of files. If you enable global resource files, you can use a single resource file (or one resource per folder) to control the screen appearance of multiple data files. This is particularly powerful when you review data stored on a read-only device such as a DVD drive.

Unlike the per-file resource files, which are updated automatically whenever you close a resource file, global resource files are never updated automatically. Use **Update Global Resource** to update the current global resource and **Save Resources As** to create a new resource file with the current settings for the current view.

The use of global resources is managed from the **Global file resources** dialog, which has these fields:



Global file resources dialog

Use global resource files

Check this box to use global resources. If this is unchecked, no global resource files are used and each data file has its own resource file. Please remember that using this dialog makes no difference to the resources used by any open time view. It changes the resource file that is used when a saved time view opens. If you check this box and no global resource file is found, Spike2 behaves as if the box were not checked.

Name of file

This field sets the name of the resource file. The name should not include a path or the file extension. These are added automatically, as required. If this field is blank, no resource file is used.

File location

This field sets where to search for the global resource file. You have three choices: *Spike2 installation folder only*, *Data file folder then Spike2 folder*, *Data file folder only*. The Spike2 installation folder is wherever the `sonview.exe` application file is located. The data file folder is the folder from which the data file is opened.

Only use global resource file if

If you do not check any boxes in this area of the dialog, global resource files are applied to all data files. This dialog area allows you to restrict the files that use the global resource file in place of the per-file resources.

Data file is within the path shown below

If you set a path and check this box, only files that lie within this path are considered for global resources. For example, if you only wanted to apply global resources to files read from your DVD drive, you might set this to `E:\` (if `E:\` is the path to your DVD).

Data file does not have its own resource file

Check this box if you would prefer to use the per-file resources if a resource file with the correct name and in the same folder as the data file exists.

Resource Files

This menu item replaces **Global Resources** when the current view is a time, result or XY view. It opens a pop-up menu from which you can select **Global Resources**, **Apply Resource File**, **Save Resources As** and **Update Global Resource**.

Apply Resource File

This command applies a user-chosen resource file to the current time, result or XY view. You can use this to apply a complex window arrangement or active cursor measurement to multiple files. The script language equivalent is `FileApplyResource()`. You can also use the **Global Resources** command to automatically apply a specific resource file.

Save Resources As

This command saves the resources associated with the current time, result or XY view to a resource file. The command opens a file save dialog for you to set the file name.

Update Global Resource

This command is enabled for the current time, result or XY view if global resources are enabled and a global resource file name is set. It updates the global resource file with the current view settings. If the global resource file does not exist, one is created in the folder set by the **Global Resources** dialog (or in the application folder if both the data file folder and the application folder are allowed).

Close, Close and Link

This command closes the active window. If you use this command on an unsaved window, you are prompted to save it before closing the window. However, if the **Edit** menu **Preferences** option is set to not prompt to save modified result and XY views, you will not be prompted to save these views and they will be lost when you close them.

Close all associated windows

If you hold down the `Ctrl` key and click on the File menu when a time view is the current window, the Close command becomes Close and Link. Select this to close the time view plus all associated windows. This does not work with a newly sampled file, save it first. If the Edit menu Preferences option is set to not prompt to save result and XY views, associated result views are deleted. However, in addition to saving the state of the data file in a `.s2rx` resource file, Spike2 also saves the state and contents of associated result view windows in the resource file. Next time you open the data file the result views are recreated from the resource file in the same state as they were closed.

Revert To Saved

You can use this command with a text file, a script file or an output sequence file. The file changes back to the state it was in at the last save.

Save and Save As

These commands are available when the current window is a text, script, output sequence, result or XY view or a time view holding newly sampled data that has not been saved or a time view that was created by the script command `FileNew(7, ...)`. **Save** writes the file with its current name unless it is unnamed, in which case you are prompted for a name. **Save As** writes the file with a different name and gives you the option of saving the data as a different type.

Time view files are kept on disk, not in memory, as they can be very large. Changes made to these files are permanent as they are made on disk. When you save a newly sampled data file, you give the file a name (replacing a temporary name). If you save it to a different drive from that set in the Edit menu Preferences, Spike2 copies the file to the new drive, then deletes the original. Moving a large file can take several seconds.

Saving an unsaved time view causes Spike2 to recalculate the file length, which determines the time range that can be displayed on the x axis. Spike2 only considers channels that have been written to disk, so memory channels and virtual channels are not taken into account.

Sequence, text, result, XY and script files are held in memory. Changes made to these files are not permanent until the file is saved to disk.

Save As for Result and XY views

In addition to saving Result and XY views in their native format, you can also choose to save them as text or as an image. The available formats are:

Text file

This is the same as the Edit menu Copy As Text command, but with the output sent to a text file and not the clipboard. This writes selected channels or all visible channels if no channel is selected.

Bitmap file, JPEG image, Portable Network Graphic file, Tagged Image Format file

These options copy the screen area containing the window to a file; the result is a copy at the screen resolution. If you need to scale the image, or want to edit it, a Metafile copy is often better. The Portable Network Graphic format will usually be the smallest on disk unless the screen contains a real-world image or a sonogram, when JPEG may be smaller (but at the cost of artefacts around axis lines and text). Bitmap images will usually be the largest.

Metafile

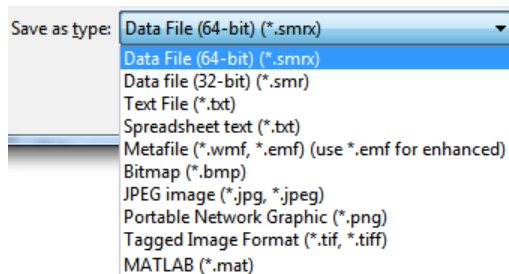
This option copies the window as a Windows (Placeable) Metafile (`*.WMF`) or as an Enhanced Metafile (`*.EMF`). Enhanced Metafiles are theoretically better as they support the cubic spline and sonogram drawing modes. However, some graphics programs do not import Enhanced Metafiles very well. The default format is a Windows Metafile. To output in enhanced format, set the file extension to `.EMF`. For example, to save to the file `fred` as a Windows Metafile, set the file name to `fred` or `fred.wmf` but to save as an enhanced metafile set the file name to `fred.emf`.

These file formats can be scaled without losing resolution and are the preferred format for moving Spike2 images to drawing programs. The Edit menu Preferences... option lets you increase (or decrease) the output

resolution. This can be important when saving time view and result view data as the number of vectors produced when drawing high resolution may stop some drawing programs from reading the file.

Export As

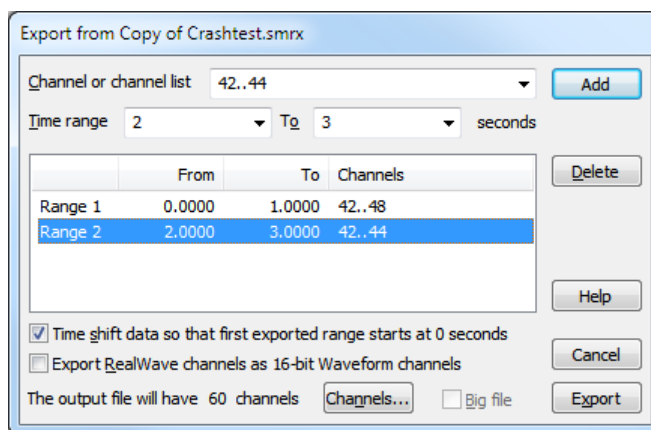
This menu item replaces **Save As** when the current window is a saved time view and opens a File Save dialog. You can choose from: **Data file (64-bit) (*.smrx)** to export as a new 64-bit Spike2 file, **Data file 32-bit (*.smr)** to export as an old-style 32-bit Spike2 data file, **Text file** or **Spreadsheet text (*.txt)**, **Metafile (*.wmf or *.emf)** for a scalable image and a variety of bitmap image formats. More types are listed if you have installed external exporters.



Choose a format and either select an existing file to overwrite or type a new file name, then click **Save**. What happens next depends of the export format:

Data file (64-bit or 32-bit)

A dialog opens in which you select channels and data sections to export. Output is to a 64-bit file unless the file name ends `.smr`, when it is to a 32-bit file. Set channels and a time range with the top section of the dialog and click **Add** to append to the list. Do this as often as required. Click **Delete** to remove a list entry. You can choose channels from the drop down list or type in a channel specification. You can export channels from the original file plus memory and virtual channels; duplicated channels are not listed. Check the **Time shift data...** box to time shift the output data so that the **Range 1 From** time becomes 0.0 in the output file.



Export RealWave channels as 16-bit Waveform channels

Check this box to convert RealWave channels (normally stored as 32-bit floating point values) to 16-bit integers using the channel scale and offset. This is usually only required when exporting as a 32-bit file so that versions of Spike2 before 4.03 can read the data. This field is omitted when external exporters use the dialog.

Channels...

This button opens a dialog in which you can set the maximum number of channels in the output file, from 32 to 400. External exporters omit this field.

Big file

This field is enabled when you export to a 32-bit file. Check the **Big file** box if you want the file written in Big file mode. As files written in this mode are not compatible with old versions of Spike2 you should not check this box unless your output file is likely to exceed 2 GB in size. If you export to a 64-bit file this field is omitted as there are no size restrictions. External exporters also omit this field.

Export

Once you have formed a list of times and channels, click **Export** to write the selected time ranges and channels to the new data file. Channels are written in order of ascending channel number. Where possible, channels are copied to the same channel number in the output file. If this is not possible, for example for a memory channel, the channel is written to the lowest numbered unused channel in the output file. If the export process takes more than a couple of seconds, a progress dialog appears and you have the option of cancelling the export.

Text file

This is the same as the **Edit** menu **Copy As Text** command, but with the output sent to a text file and not the

clipboard. This writes selected channels or all visible channels if no channel is selected.

Spreadsheet text file

Copy a time view as a text file for easy import to a spreadsheet. This is the same as the Edit menu Copy Spreadsheet command, but with the output sent to a file. This writes selected channels or all visible channels if no channel is selected.

Bitmap, JPEG image, Portable Network Graphic, Tagged Image Format file

These options copy the screen area containing the window to a file at the screen resolution. If you need to scale the image, or want to edit it, a Metafile copy is often better. The Portable Network Graphic format will usually be the smallest on disk unless the screen contains a real-world image or a sonogram, when JPEG may be smaller (but at the cost of artefacts around axis lines and text). Bitmap images will usually be the largest.

Metafile

This option copies the window as a Windows (Placeable) Metafile (*.WMF) or as an Enhanced Metafile (*.EMF). Enhanced Metafiles are theoretically better as they support the cubic spline and sonogram drawing modes. However, some graphics programs do not import Enhanced Metafiles very well. The default format is a Windows Metafile. To output in enhanced format, set the file extension to .EMF. For example, to save to the file `fred` as a Windows Metafile, set the file name to `fred` or `fred.wmf` but to save as an enhanced metafile set the file name to `fred.emf`.

These file formats can be scaled without losing resolution and are the preferred format for moving Spike2 images to drawing programs. The Edit menu Preferences... option lets you increase (or decrease) the output resolution. This can be important when saving time view and result view data as the number of vectors produced when drawing high resolution may stop some drawing programs from reading the file.

External exporters

These use the same dialog as for exporting to a data file to select channels and a single time range. There may be additional dialogs depending on the target format. See the `EXPORT` folder for additional printed documentation for external exporters or here in the help system. The only external exporter currently defined is for MatLab files.

MATLAB external exporter

You can export a time view, result view or an XY view to a MATLAB file. Choose MATLAB data as the Save As type in the Export As dialog in a time view or in the Save As dialog for result and XY view. You can also control export to MATLAB format from a script.

Working with large files

The writing to MATLAB files is done by calls into MATLAB-supplied routines. There are limits on the size of MATLAB objects that can be written. Use format version 7.3 for huge files (beware that they are compressed, which slows export). You need a 64-bit MATLAB to read gigabyte sized files. If you intend to export large files you must be realistic about how long it may take (several minutes).

Workspace variable naming

The workspace variables in a mat-file must have names that are both unique and legal. If a variable name is not unique it will overwrite the workspace data with the same name. MATLAB cannot read variables with illegal names. Spike2 mat-file export creates variables with names based upon the source view name and the channel name. You can independently select if either of these is to be used. This produces variable names that are both useful and unique. The default setting is to use the source view name but not the channel name. The variable name settings are used to build the variable name as follows:

If the source view name is to be used, it is placed at the start of the variable name followed by an underscore character `'_'`. Following this a channel identifier is added. This is either the channel title (if using the source channel name is selected and the name is not blank) or Spike2 builds a name using `'ch'` followed by the

channel number. For example, when exporting a view (of any type) called `Expt1` containing 3 channels; channel 1 called 'ECG', channel 3 called 'AP' and channel 7 with a blank name we would get the following MATLAB variable names:

Using source view and channel name: `Expt1_ECG, Expt1_AP, Expt1_Ch7`
 Using source view name only: `Expt1_Ch1, Expt1_Ch3, Expt1_Ch7`
 Using channel name only: `ECG, AP, Ch7`
 Using neither name: `Ch1, Ch3, Ch7`

Having generated a name using these rules, Spike2 checks and, if necessary, modifies the name to guarantee that it is legal. The rules for a legal name are that it must not be more than 63 characters, must begin with an alphabetic character and must only contain alphanumeric characters and the underscore character '_'. Spike2 modifies the name by appending a 'v' to the start of the name if it does not begin with an alphabetic character, converting all illegal characters to underscores and finally truncating if more than 63 characters long.

WARNING: If you end up with variable names that are the same, the variables will overwrite each other when you read the file into MATLAB and you will only see the last read variable.

To export data as a MATLAB file, open the File menu **Export As** command from a time view or the **Save As** command from a result or XY view and select **MATLAB data (*.mat)** in the **Save as type** drop-down list. Set a file name and click **Save** to proceed to the next dialog where you choose the data to export and the format to export it in. The details of the new dialog depend on the source view type:

Time view data

When exporting time view data, you first get a dialog that allows you to select the channels to export and the time range that you want to export over. There is also a check box that selects offsetting the start of the time range being exported to zero. This dialog is a simplified version of the dialog used to select data to export to a SON file, however you can only select a single set of channels and time range.

Note that, when exporting waveform data to a mat-file, only the first contiguous section of waveform data within the selected time range will be exported. The export data format cannot manage more than one section of contiguous waveform data.

When you have selected your channels and time range you are then presented with a second dialog that controls mat-file export options. These options allow you to set which MATLAB file format to use, how the MATLAB workspace variables will be named, what data is generated for different channel types and the format used for various types of Spike2 data. These options are also available when you export using a script.

Compatibility

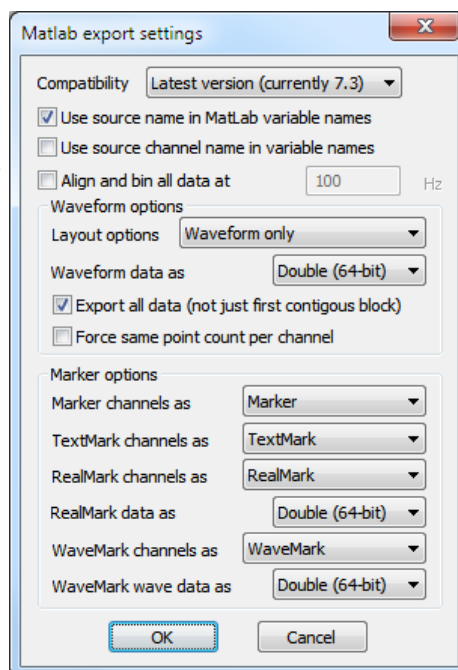
If you are using an older version of MATLAB, you must use the compatibility field to select a suitable output file format format. Version 7 and 7.3 files are compressed, which can be slow.

Use ... in variable names

These two options in the dialog control how the mat-file variable names are constructed as described previously.

Align and bin all data at

If you check this box, the field to the right of the text is enabled and you can set the frequency at which the output data is generated on all channels. Waveform data and RealMark data draw as a waveform are re-sampled to this frequency using cubic spline interpolation and all other event and marker data is converted to binned event counts using the specified frequency to set the bin width. The options for event and marker channels also change; this is described below.



Waveform options

These controls specify how Waveform and RealWave data is handled.

Layout options

The Layout options item can be set to **Waveform only** or **Waveform and times**, if the second option is selected then an array containing all of the sample times is created as well as the array of waveform values.

Waveform data as

The Waveform data as item sets the type of data created for waveforms, it can be set to **Integer (16-bit)**, **Float (32-bit)** or **Double (64-bit)**. Integers use least memory (but need scaling to user units), float has the full accuracy of the data and is recommended. If you choose Integer for RealWave data, the values are converted to integer using the channel scale and offset (limited to the range -32768 to 32767).

Export all data (not just first contiguous block)

Waveforms channels in Spike2 data files can contain gaps. Normally, data export stops when a gap is encountered. If you check this option, the export ignores the gaps. If you select this option and your data does contain gaps, you will probably want to set the layout option to **Waveform and times** so that you can detect where the gaps are.

Force same point count per channel

This option is available when all the waveform channels have the same sample rate. Unless all the channels are aligned to the same sample time, in an arbitrary time range some of the channels will have n data points and some will have $n+1$. If you check this option, the end time for the data export is adjusted forwards in time by up to one sample period so that all the waveform channels will have the same number of points. This can only be successful if the waveform data on all channels is contiguous (no gaps) over the time range set for export and the end time does not hit the last data point on any channel.

Marker options

The rest of the dialog sets how the various types of marker data are handled. The controls labelled **Marker channels as**, **TextMark channels as**, **RealMark channels as** and **WaveMark channels as** define what data is generated for the specified type of channel. If these items are set to **Event** then the channels are treated as events and a single array of times is generated. If they are set to **Marker** then an additional array containing the marker codes is generated, and (for the extended marker types only), setting them to **TextMark**, **RealMark** or **WaveMark** means that another array containing the additional marker information is also created.

The remaining two controls labelled **RealMark data as** and **WaveMark wave data as** set the type of data produced for these channel types, RealMark data can be exported as 32-bit floats or 64-bit doubles while WaveMark wave data can be exported as 16-bit integers as well in the same manner as Waveform data.

If you check the **Align and bin all data** at check box, the options for markers are replaced by a single check box labelled **Generate times for binned data**. Check this to generate an array of bin times as well as the bin counts in the same way as for the Layout options item for waveform data.

Error handling

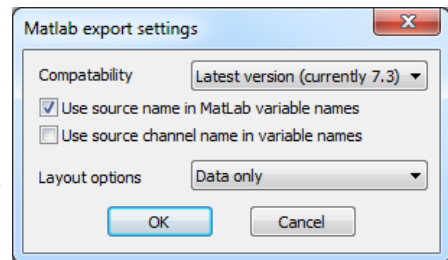
If there is an error in the underlying channel data (for example, if the data file associated with a time view is damaged), the damaged channel will not be exported and you will get an error message. You can recover damaged `smr` files with SonFix and damaged `smrx` files with S64Fix.

Result view data

When exporting a result view to a mat-file you are first presented with a simple dialog to select the X axis range of the data to be exported.

This is followed by a second dialog that sets mat-file export options. Use the **Compatibility** field to select the file format to use. Unless you have an older version of MATLAB you should select the **Latest** output format.

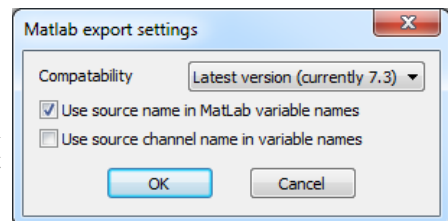
The next two controls are check boxes to control the channel variable naming as described above. The only other control, labelled **Layout options**, can be set to **Data only** if you only want to export the result bin values or **Data and times** if you want to export both the bin values and the bin x axis positions.



XY view data

For an XY view you do not get a dialog to control what data is exported; all the visible channels are exported and only data points that lie within the displayed X and Y axis ranges are used.

A dialog is provided to control how the data is exported. This is a simplified version of the result view export options dialog; it contains the two standard check boxes to select the use of the source view name and channel names to create the MATLAB variable name plus the choice of which version of the file system to use.



Mat file data format

A mat-file represents a collection of MATLAB variables rather than simple data values, which allows for a complex representation of Spike2 data within MATLAB. Each Spike2 channel exported into a mat-file is represented by one variable. This variable is a structure containing fields that hold information about the channel and the channel data. For example, a structure holding data from a waveform channel has the fields `title`, `frequency` and `start` each of which is a simple (scalar) datum holding the channel title, the sampling frequency in Hz and the time of the first data point. In addition there is a field called `wave` that is a $n \times 1$ matrix holding the waveform data values and an optional field called `times` which is a $n \times 1$ matrix holding the sample time for each of n waveform points. The structure varies according to the channel type, though some fields are common to all channels.

Waveform RealWave and binned RealMark data drawn as a waveform

Waveform data is exported as a 1-dimensional array of waveform values with an optional associated array of sample times. The fields in the channel structure are:

<code>title</code>	a string holding the channel title.
<code>comment</code>	a string holding the channel comment.
<code>interval</code>	a double holding the sample interval or bin width in seconds.
<code>scale</code>	a double holding the channel scale factor. The scale and offset values convert 16-bit ADC data to real values using the equation: $real = (ADC * scale) + offset.$
<code>offset</code>	a double holding the channel <code>offset</code> value (see above).
<code>units</code>	a string holding the channel units.
<code>start</code>	a double holding the time of the first waveform point in seconds
<code>length</code>	a double holding the number of waveform items.

values	a length x 1 array holding the waveform values, either raw or produced by interpolating to the bin times. Depending upon the output options selected, this array could be double or single-precision real values or 16-bit integers.
times	a length x 1 array of doubles holding the sample times of the waveform values in seconds. This field is only present if waveform times are selected.

If you choose to export a RealWave channel as 16-bit integers, the Spike2 channel scale and offset (as seen by double clicking the channel title in Spike2 or accessed from the Spike2 script language with the `ChanScale()` and `ChanOffset()` commands) are used to convert real values into integers. If the result exceeds the 16-bit signed range (-32786 to 32767), the result will wrap around.

Event channels binned and un-binned

Event data is represented as a 1-dimensional array of event times with (for Level channels) a corresponding array of levels. If the data is binned, this is replaced by an array of bin counts and an optional array of bin times. This format is used for all types of marker channel if they are exported as event data as well as for event channels. If the data has been binned, this format is used for all types of marker. The fields in the channel structure for both types of data are:

title	a string holding the channel title.
comment	a string holding the channel comment.
length	a double holding the number of events or binned values.

For un-binned data, the extra fields are

resolution	a double holding the underlying timing resolution in seconds.
times	a length x 1 array of doubles holding the event times in seconds.
level	a length x 1 array of byte values holding the level for level channels only. The value is 1 if the transition at the corresponding time was upwards, 0 for a downwards transition

For binned event and marker data these fields are replaced by:

interval	a double holding the bin width in seconds.
start	a double holding the time of the first bin in seconds.
values	a length x 1 array of doubles holding the binned event counts.
times	a length x 1 array of doubles holding the times in seconds for the corresponding bins. This field is present if bin times are selected.

Marker channels

Marker data is represented in the same way as simple event channels but with the addition of an extra length-by-4 byte array of marker codes. This format is used for all types of extended marker channel if they are exported as markers. The fields in the channel structure are:

title	a string holding the channel title.
comment	a string holding the channel comment.
resolution	a double holding the underlying timing resolution in seconds.
length	a double holding the number of markers.
times	a length x 1 array of doubles holding the marker times in seconds.
codes	a length x 4 array of byte values holding the marker codes.

TextMark channels

TextMark data is represented in much the same way as simple markers but with the addition of an extra 2 dimensional array holding the marker text. The fields in the channel structure are:

title	a string holding the channel title.
comment	a string holding the channel comment.

resolution	a double holding the underlying timing resolution in seconds.
length	a double holding the number of markers.
items	a double holding the maximum number of characters in each marker.
times	a length \times 1 array of doubles holding the marker times in seconds.
codes	a length \times 4 array of byte values holding the marker codes.
text	a length \times items array of char values holding the marker text.

RealMark channels

RealMark data is represented similarly to simple markers but with the addition of an extra 2 dimensional array holding the real values and units information. The fields in the channel structure are:

title	a string holding the channel title.
comment	a string holding the channel comment.
resolution	a double holding the underlying timing resolution in seconds.
units	a string holding the channel units.
length	a double holding the number of markers.
items	a double holding the number of real values per marker.
times	a length \times 1 array of doubles holding the marker times in seconds.
codes	a length \times 4 array of byte values holding the marker codes.
values	a length \times items array holding the marker values. These are either single or double-precision real values, set by the export option selected.

WaveMark channels

WaveMark data is represented in the same way as simple markers but with the addition of an extra 3 dimensional array holding the waveform values and additional calibration information as provided for waveform values. The fields in the channel structure are:

title	a string holding the channel title.
comment	a string holding the channel comment.
resolution	a double holding the underlying timing resolution in seconds.
interval	a double holding the waveform sample interval in seconds.
scale	a double holding the waveform scale factor. The scale and offset values convert 16-bit ADC data to real values using the equation: $real = (ADC * scale) + offset.$
offset	a double holding the waveform offset value (see above).
units	a string holding the waveform data units.
length	a double holding the number of markers.
items	a double holding the number of waveform values per trace per marker.
trigger	a double holding the offset to the trigger point within a trace in the waveform data. This value will be from 0 to items-1;
traces	a double holding the number of traces within the waveform data (normally the number of electrodes).
times	a length \times 1 array of doubles holding the marker times in seconds.
codes	a length \times 4 array of byte values holding the marker codes.
values	a 3-dimensional array (length \times items \times traces) holding the marker waveforms. For single trace data the third dimension is 1 so it becomes a length \times items matrix. Depending upon the output options selected, this array could be double or single-precision real values or 16-bit integers.

Result channels

Result data is represented in much the same way as waveform data but without any calibration information. Extra information is provided about the X axis. The fields in the channel structure are:

<code>title</code>	a string holding the channel title.
<code>units</code>	a string holding the channel (Y axis) units.
<code>xunits</code>	a string holding the X axis units.
<code>interval</code>	a double holding the result bin width in the appropriate X axis units.
<code>start</code>	a double holding the X axis value of the first bin.
<code>length</code>	a double holding the number of result bins.
<code>values</code>	a <code>length x 1</code> array of doubles holding the result bin values.
<code>times</code>	a <code>length x 1</code> array holding the X axis values for the corresponding bins (these are not always time values). This field is only present if result bin times are selected.

XY channels

XY data is represented as two 1-dimensional arrays of X and Y data plus ancillary information. The fields in the channel structure are:

<code>title</code>	a string holding the channel title.
<code>units</code>	a string holding the Y value units.
<code>xunits</code>	a string holding the X value units.
<code>length</code>	a double holding the number of XY points.
<code>xvalues</code>	a <code>length x 1</code> array of doubles holding the x values of the data.
<code>yvalues</code>	a <code>length x 1</code> array of doubles holding the y values of the data.

Script export to mat files

Export to mat-files is also available from the script language by using the `FileSaveAs()` function with the file type set to 100. With this type in use, the file name selection dialog (if provided) will use a `*.mat` filename filter, an extra `exp$` argument becomes available to allow MATLAB-export specific options to be set and the time range and channels to be exported are set by `ExportChanList()`. Only the first set of time range plus channels is used, and `ExportChanList()` is additive in action, so you should first use `ExportChanList()` with zero (or one) arguments to clear out any stored sets of information before setting the time range and channels you want. The extra `exp$` argument that sets options is a string of the form:

```
name=value|name=value|...|name=value
```

where `name` specifies some export option and `value` sets it's value. You can include as many options as you want, options that you omit are set to the default value. Option names are not case-sensitive. It is not an error to use an unknown option.

BEWARE: If you set `UseCName` to 1 and you have channels with duplicated names, Spike2 will write the data, but MATLAB will only see the last channel as each variable with the same name overwrites the previous one when data is read.

Time view export options

<code>UseSName</code>	selects use of the source name in the channel variable name. Set 1 (default) to use the source name, 0 if not.
<code>UseCName</code>	selects use of the channel name in the channel variable name. Set to use the channel name, 0 if not. The default is 0.
<code>BinFreq</code>	selects binning of the data. Set 0 (default) for unbinned, otherwise set to the binning frequency in Hz.
<code>BinTimes</code>	selects generation of an array of the times of the bins (ignored if the data is not being

	binned). Set 1 for bin times, 0 if not. The default is 0.
WaveTimes	selects generation of an array of waveform sample times in the channel variable. Set 1 for times, 0 (default) for none.
WaveData	selects the data type generated for waveform channels. 0 for 16-bit integers, 1 for single-precision reals and 2 for double-precision reals. The default is 2.
WaveAll	Set to 0 to stop waveform or RealWave export at a gap. Set to 1 to ignore gaps. The default is 0.
WaveSameN	Used when all waveform channels have the same sample rate. Set to 1 to adjust the end of the time range so that all channels have the same number of points. Set to 0 (default) to use the time range as supplied.
MarkAs	sets how marker data is exported. Set to 0 for events and 1 for markers. The default is 1.
TMarkAs	sets how text marker data is exported. Set to 0 for events, 1 for markers and 2 for text markers. The default is 2.
RMarkAs	sets how real marker data is exported. Set to 0 for events, 1 for markers and 2 for real markers. The default is 2.
RMarkData	selects the data type generated for the real values in RealMark channels. 1 for single-precision reals and 2 for double-precision reals. The default is 2.
WaveMarkAs	sets how wave marker data is exported. Set to 0 for events, 1 for markers and 2 for wave markers. The default is 2.
WaveMarkData	selects the sort of data generated for the waveform in WaveMark channels. Set to 0 for 16-bit integers, 1 for single-precision reals and 2 for double-precision reals. The default is 2.

Result view export options

UseSName	selects use of the source name in the channel variable name. Set to 1 if you want to use the source name, 0 if not. The default is 1.
UseCName	selects use of the channel name in the channel variable name. Set to 1 if you want to use the channel name, 0 if not. The default is 0.
BinTimes	selects generation of an array of result bin times in the channel variable. Set to 1 if you want bin times, 0 if not. The default is 0.

XY view export options

UseSName	selects use of the source name in the channel variable name. Set to 1 if you want to use the source name, 0 if not. The default is 1.
UseCName	selects use of the channel name in the channel variable name. Set to 1 if you want to use the channel name, 0 if not. The default is 0.

Compatibility

Compat	Sets which version of the MATLAB filing system to support. 0 (default) for the most recent MATLAB file format known to the system, 1 for version 4 and earlier, 2 for version 6, 3 for version 7 and 4 for version 7.3 (supports files > 4 GB).
--------	---

Load and Save Configuration

These commands manage Spike2 configuration files (.s2cx extension). Configuration files hold the full sampling setup including any output sequence file, waveform output and window arrangement and the types of on-line analysis required. The Load Configuration and Save Configuration commands transfer the Spike2 configuration between disk and the application. They both open an appropriate file dialog to select a file for loading or saving. You cannot load a configuration file if you are sampling data.

You can read some sampling parameters from a Spike2 data file (.smr extension); this is useful if you have lost the corresponding configuration file. Data files hold only the channel settings; they do not hold window arrangements, output sequencer, waveform output or data processing settings. Select Data files (*.smr) in the dialog to read a data file in place of a configuration file. **There is no guarantee that a configuration read in this way will exactly match the data file**; you must check the result carefully and you may need to adjust the resulting configuration.

After reading a configuration, the Sampling Configuration dialog opens to display it. If you wish to sample immediately, click the Run Now button.

You can also click a button in the Sample Bar to load a configuration and start sampling.

Default configuration files: default.s2cx, last.s2cx

If the configuration file `default.s2cx` is found, it is loaded when Spike2 starts. To save this file, hold down the control key while activating the File menu and use the Save Default Configuration command. The standard file `last.s2cx` holds the last configuration that was used for sampling. If `default.s2cx` cannot be found, and `last.s2cx` exists, `last.s2cx` is loaded. Spike2 saves `last.s2cx` each time sampling stops.

Where does Spike2 search for these files?

Now that we offer the option of installing Spike2 in the Program Files folder, the places where modifiable files can be stored has changed. As we want to remain compatible with the old system, we have to search a list of places to find the `default` and `last` files. The search order is:

1. The application data directory (but read below for what this means)
2. The directory in which Spike2 is installed

In each directory we search for the `s2cx` file first, then for the `s2c` file (for backwards compatibility). The application data directory is the first of the following that exists:

1. The current users application data directory
2. The all-users application data directory
3. The "My Documents" folder

Double-click .s2c files to open in Spike2

If double-clicking a `.s2cx` file in the shell (desktop or a file folder window) does not open the file in Spike2, you should check that the file association are set correctly. In Windows XP:

1. Open a file folder and in the folder Tools menu, select Folder Options...
2. Select the File Types tab, click on a listed file extension and type `s` to scroll to near the `s2cx` file entry
3. Scroll down the list to find `s2cx CED Spike2 configuration file` and select this item
4. Click the Advanced button, then on the **open** action, then click Edit...
5. Check that the Application used to perform action points at Spike2 and ends with `sonview.exe "%1"`
6. Check that Use DDE is checked and DDE Message is set to `[open("%1")]`
7. Check that Application is set to SONVIEW
8. DDE Application Not Running should be blank
9. Topic should be set to System
10. Accept any changes by clicking OK to get back to the File Types dialog, then close the dialog.

In Windows 7:

1. Click the Start button, then click Default Programs
2. Click Associate a file type or protocol with a program
3. Locate `.s2cx` and select it, then select Change program...
4. Locate `sonview.exe` and select it, then close the default program windows
5. Double-clicking the file will now run Spike2 and open the file as long as Spike2 is not already running.

Exit

This command closes all open files and exits from Spike2. If there are any text or output sequence files open that have not been saved, you will be prompted to save them before the application terminates.

Send Mail

If your system has support for Mail installed (for example Microsoft Exchange), you can send documents from Spike2 to another linked computer. This option vanishes if you do not have compatible mail support.

Text-based documents and result views can be sent, even if they have not been saved to disk (Spike2 writes them to a temporary file if they have not been saved).

You can send a Spike2 data file, but only if you are not sampling to the file and it has been saved on disk and is unmodified. Spike2 makes a temporary copy of the file in the system temporary folder before mailing it to avoid problems with mail programs that will not send a file if it is open in another application. You must have enough spare disk space for at least 2 copies of the data file.

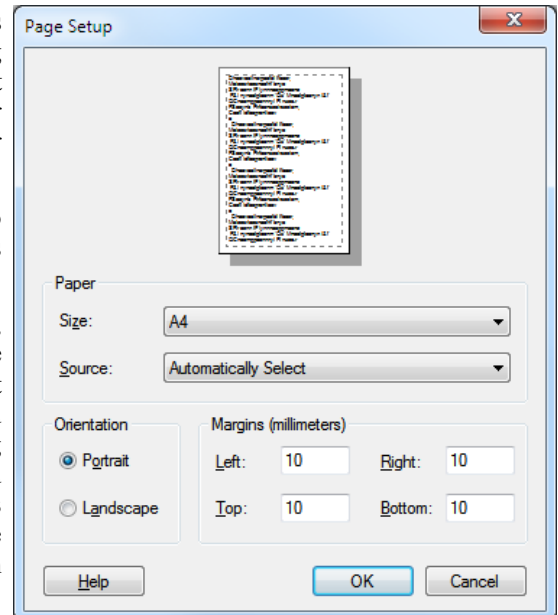
Printing

Page Setup

This opens the printer page setup dialog. The dialog varies between operating systems and printers. See your operating system documentation for more information. The important options that are always present include the paper orientation (portrait or landscape), the paper source (if your printer has a choice), and the printer margins.

The Orientation option (portrait or landscape) applies to all output except the Print Screen option, which has its own selector for the output orientation.

The printer margins will appear in inches or in millimetres, depending on the locale set for your computer. These margins are used for all printed output. The left and right margins are applied to everything, including headers and footers. The top and bottom margins apply to everything except headers and footers, which have their own top and bottom margin settings (see the Page Headers description). The top and bottom margins you set here are further modified if a header or footer would collide with them.



Most printers have an unprintable area near the paper edge. If you set margins smaller than the unprintable area, the margins are increased so that all your output is visible. If you set margins that reduce the printable area too much, the margins are ignored.

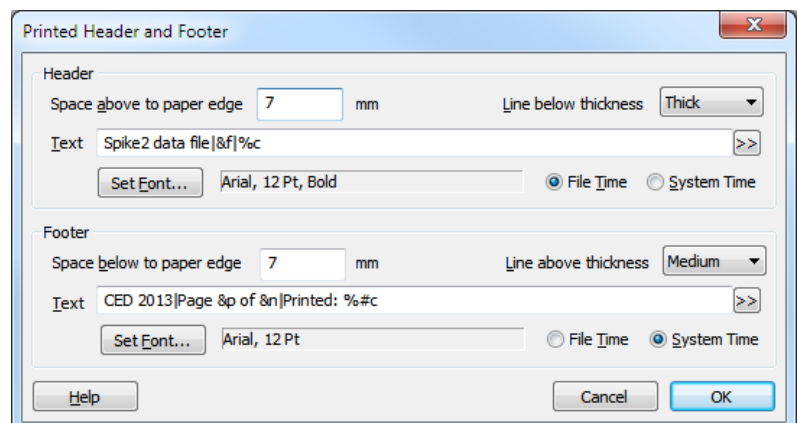
Registry use

Spike2 saves the page margins in units of 0.01 mm in the system registry. You can find them in the HKEY_CURRENT_USER\Software\CED\Spike2\PageSetup folder as REG_DWORD values: LeftMargin, RightMargin, TopMargin and BottomMargin.

Page Headers and Footers

You can apply headers and footers to all printed output. The headers set in this dialog are used with all Time, Result, XY and text-based views. The header and footer text is displayed in the page format set by the Page Setup command.

Other printed output (for example from the Print Screen command) uses all these settings except the header and footer text, which is provided by each of the other printing commands.



Page Header and Footer dialog

Header and footer positions

The horizontal position is set by the left and right margins in the Page Setup dialog. The vertical positions are set by the space above the header and the space below the footer fields in inches or millimetres, depending on the locale. If your header or footer encroaches on the top and bottom margins set in the Page Setup dialog, the top and bottom values are adjusted to keep the output clear of the header and footer.

Line thickness

You can choose between: None, Hairline, Thin, Medium and Thick. A Hairline is the thinnest line possible on the output device. The other settings should be self-explanatory. The header line is drawn below any header text, the footer line is drawn above any footer text. If there is no header or footer text, no line is drawn.

Text

This is the text to display as the header or footer. If this field is empty, the header or footer is omitted (including any line). You can split the text into left-justified, centred and right-justified sections with the vertical bar character. You can also insert codes that are replaced by document and time information. The simplest way to do this is by clicking the >> button to the right of the text and choosing an option.

Set Font

Click this button to choose a font for the header and the footer. Font sizes are limited to 2 to 30 points.

File/System Time

You can insert times into both the header and the footer. However, you have to choose between the current time and the file time. This allows you to display the file time in the header and the current time in the footer, or vice versa.

Document information

The following codes are replaced by document information:

&f	File and path	&F	Upper case file and path
&t	Document title	&T	Upper case document title
&p	Page number	&n	Total number of pages
&&	The ampersand sign (&)		

Time and date codes

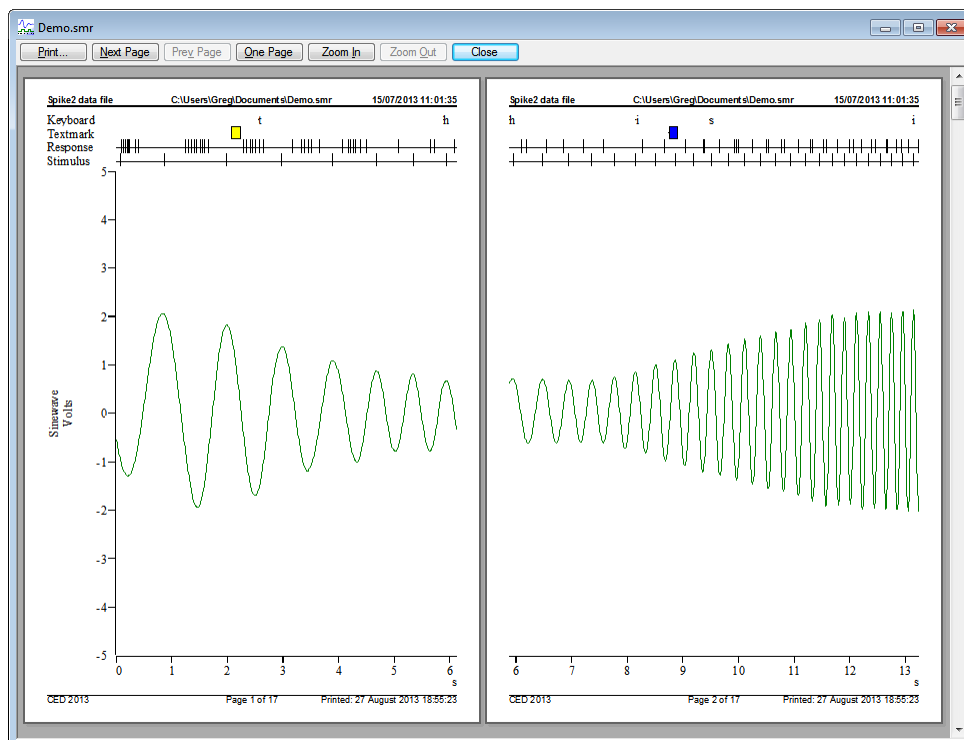
You can insert times and dates using % followed by a character code. The combination is replaced as described in the table. You can also use %#c and %#x for a long version of dates and times. You can remove leading zeros from numbers with #, for example %#j.

%a	Short day of week (Mon)	%P	Indicator for A.M or P.M.
%A	Long day of week (Monday)	%S	Seconds as number (00-59)
%b	Short month name (Jan)	%U	Week of year, Sunday based (00-53)
%B	Long month name (January)	%W	Week of year, Monday based (00-53)
%c	Date and time for locale	%w	Sunday based weekday (0-6)
%d	Day of month (01-31)	%x	Date formatted for locale
%H	Hour in 24 hour format (00-23)	%X	Time formatted for locale
%I	Hour in 12 hour format (01-12)	%Y	Year without century (00-99)
%j	Day of year (001-366)	%Y	Year with century, e.g. 2004
%m	Month as number (01-12)	%z/Z	Time zone name
%M	Minute as number (00-59)	%%	The percent sign (%)

Registry use

Spike2 saves the header and footer settings in the system registry. You can find them in the folder HKEY_CURRENT_USER\Software\CED\Spike2\PageSetup as strings: Header, Header info, Footer and Footer info. The Header and Footer items hold the text strings. The two info items are strings that code up the font name, point size, bold and italic settings, distance to the paper edge in units of 0.01 mm, line thickness (0=none, 1=Hairline, 2=Thin, 3=Medium, 4=Thick), and File time (0) or System time (1).

Print Preview



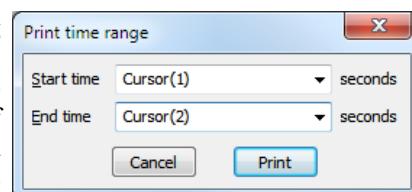
This option displays the current time, result, XY and text-based window as it would be printed by the Print option. You can zoom in and out, view one or two pages, step through pages of multi-page documents and print the entire document using a toolbar at the top of the screen. **Close** leaves this mode without printing.

The preview takes place inside the frame of the time, result or XY view. You can access the File menu to change the headers and footers and print margins.

Print Visible Print and Print Selection

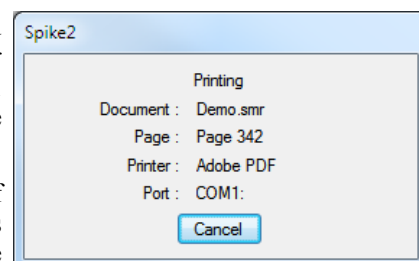
These commands print time, result, XY and text-based windows. Any scroll bar at the bottom of the window is not printed. The **Edit** menu **Preferences** dialog sets the line widths used on screen and for printing. **Print Selection** prints the selected area of a text window. **Print Visible** prints the visible data in the current window. **Print** prints a specified region of a time, text or result view; each printed page holds the x axis range of the window. **Print** in an XY view is the same as **Print Visible**.

To print an entire data file, set the width of the time window holding the file to the page size required in the print, then select **Print**. Set the start time to zero and use the drop down list to set the end time to **Maxtime()**. Beware that **Print** could require several hundred miles of paper to complete the printing job in the worst case! If you displayed 10 milliseconds of data across the screen in a file that is 1000 seconds long, there are 100,000 pages to print.



Both commands open the standard print dialog for your printer. You can set the print quality (in general, the better the quality, the longer the print takes) and you can also go to the Setup page for the printer. Once you have set the desired values, click the **Print** button to continue or the **Cancel** button to abandon the print operation.

During the print operation (which can take some time, particularly if you selected a lot of data) a dialog box appears. If your output spans several pages, the dialog box indicates the number of pages and the current page, so you can gauge progress. If you decide that you didn't want to print, click the **Cancel** button.



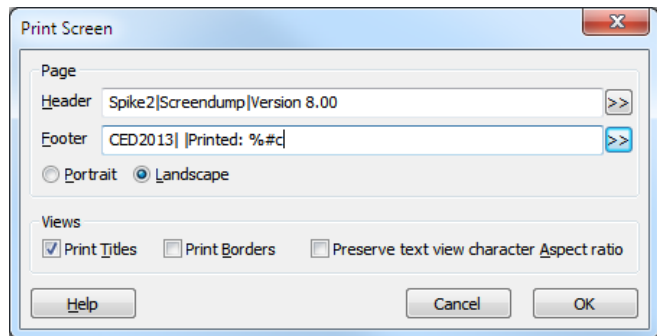
Print Screen

The Print Screen command prints all time, result, XY and text based views to one printer page. The views are scaled to occupy the same proportional positions on the printed page as they do on the screen. The page margins are those set by the Page Setup and Page Headers dialogs.

The command opens a dialog with two regions: Page and Views. In the Page region you can set a page header and footer and choose to print in Landscape or Portrait mode. The header text is printed with an underline across the width of the page. The footer text is printed with a line over it across the page width. The fonts used and line thicknesses are as set in the Page Headers dialog. If the header or footer is blank, both it and the associated line are omitted.

You can divide the header and footer into a left justified, centred and right justified sections with the vertical bar character, for example: Left|Centered|Right. You can include the current date and time in the header or footer by including, for example %c, as described for the Page Headers dialog. The >> button can be used to insert the time and date and vertical bar separators into the header and footer.

In the Views region you can choose to print view titles and draw a box round each view. You can also ask Spike2 to attempt to preserve the aspect ratio of characters in text windows. This feature is currently not supported.



Registry use

Spike2 saves the Print Screen settings in the system registry. You can find them in the HKEY_CURRENT_USER \Software\CED\Spike2\PageSetup folder. The text strings are saved as PSHeader and PSFooter. The remaining values are saved as REG_DWORD values of 0 (not selected) and 1(selected): PSViewTitle, PSBorder, PSScaleText and PSLandscape.

7: Edit menu

Edit menu

This menu holds the standard Edit functions that all programs provide. The majority of the menu is associated with commands that move data to and from the clipboard.

Undo and Redo

In a text, script or output sequence window this is used to undo or redo the last text edit operation. You cannot undo operations that have been saved to disk or text operations that were done by a script.

In Time, Result and XY windows, you can undo most operations that change the appearance of the window. Cursor movements do not undo; this is by design so you can zoom in, adjust a cursor, then undo to zoom out, leaving the cursor adjusted.

Cut

You can cut editable text to the clipboard in any position in Spike2 where the text pointer is visible. You cannot use this command in Spike2 data document windows or in result windows.

Copy

You can copy editable text to the clipboard plus selected fields from the **Cursor Regions** and **Cursor Values** dialogs. If you use this command in a time, result or XY window, the contents of the window, less the scroll bar, are copied to the clipboard as both a bitmap and as a metafile. See also **Copy As Text**.

Metafile output

To export an image to a drawing program for further annotation and manipulation, you can paste the image as either a bitmap or as a metafile. Metafiles are usually the preferred choice as you can treat the image as lines and text for further work. You can set the metafile scaling and if the image is saved as a Windows Metafile or as an enhanced Metafile in the **Edit menu Preferences**.

Copy Spreadsheet

This copies selected time view data channels to the clipboard as text. Use **Export As** to copy to a text file. If there are no selected channels, it copies visible channels. The data is written in columns. The first column holds the time of each row (in seconds). The remaining columns hold the data, one column per channel. The first line holds titles to identify the data.

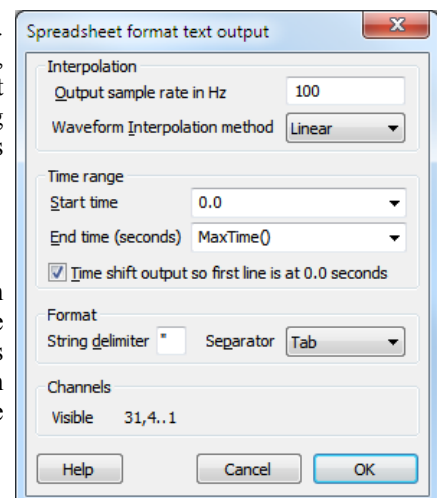
Channel output

Where possible, the output matches the display. Channels drawn with a y axis output values that match the display. Channels drawn in State mode output the state code as text. Level event channels drawn as levels output 1 or 0. Sonogram draw mode is output as waveform mode. All other drawing modes output the number of events from the time of the current line up to the time of the next line.

Interpolation

Spike2 allows each channel to have independent rates and types. To make Spike2 data easily accessible to other programs we resample the data to a common rate, set by the **Output sample rate in Hz** field.

In general, the sample rates of Waveform, RealWave and WaveMark data will not match the output rate you



have chosen and the program must interpolate. You can choose between Linear, Cubic spline interpolation, or use the value at the Nearest data point. Waveform data drawn in Sonogram mode is treated as a waveform. All other channels that need interpolation use linear interpolation.

Time range

The Start time and End time fields set the time range to copy. You can also shift the data so that the first line has a time of 0.0 to make comparisons of time ranges easier.

Format

Values are written in numeric or text columns. There is a separator between columns; choose from Tab, space or comma in the Separator drop down list. The String delimiter character is placed around text output.

Copy As Text Result view

This command is available in result views. It copies the bin values in the window for all visible channels to the clipboard as text. To copy all bins, double click the x axis and select Show All, then copy. The first output column holds the x value at the left of each bin. Each channel contributes one column of bin contents plus a column of error bar sizes if error bars are enabled and a column of bin counts if bin counts are enabled. The first line holds column titles. This command does not copy channels drawn as raster data. The first example is from a result view holding two waveform averages:

```
"Time" "Filtered" "Sinewave"
0      0.84302088  3.2980477
0.01   0.27032173  2.2613753
0.02   -0.45484864  1.043198
0.03   -0.74969506  0.63429488
```

The columns are separate by Tab characters. The second example is from the same data with error bars enabled and drawn in SEM mode.

```
"Time" "Filtered" "SEM" "Sinewave" "SEM"
0      0.84302088  0.038987886  3.2980477  0.032728175
0.01   0.27032173  0.021425654  2.2613753  0.036360926
0.02   -0.45484864  0.027970654  1.043198  0.056437311
0.03   -0.74969506  0.037386934  0.63429488  0.050863126
```

Copy As Text XY view

This menu item is available in XY views. It copies the visible points for visible channels to the clipboard. If the view has one channel or the Measurement system created it with the All channels use same X option, the output is a rectangular table with the first line holding column titles. There is one column of x values followed by one column per channel for each y value. The columns are separated by a tab character:

```
"X" "Channel 1" "Channel 2"
0   0.244140625  0.0170616301
2   3.122558594  0.0001053187043
4   2.211914063  0.0006319122258
6   1.889648438  -0.0005265935215
8   0.8642578125 -0.0005265935215
```

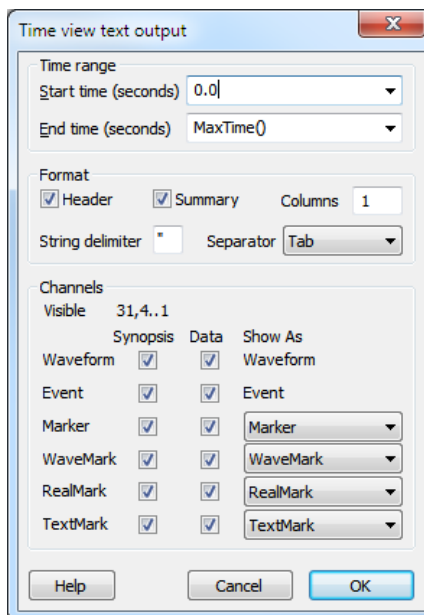
In all other cases, channels are output separately. For each channel, the first output line holds "Channel : cc : nn" where cc is the channel number and nn is the number of data points. The data points are output, one per line as the x value followed by the x value, separated by a Tab character.

```
Channel : 1 : 5
0.0170616301    0.244140625
0.0001053187043  3.122558594
0.0006319122258  2.211914063
-0.0005265935215  1.889648438
-0.0005265935215  0.8642578125
Channel : 2 : 5
0              0.0170616301
2              0.0001053187043
4              0.0006319122258
6              -0.0005265935215
8              -0.0005265935215
```

Copy As Text Time view

In a time view the command opens a dialog in which you specify the time range of data to copy, how you want the data to be copied and the output format. Select the channels to copy in the time view before you use this option. If you select no channels, Spike2 copies visible channels. Text representations of sampled data can be very large and awkward to manipulate with the clipboard. Alternatively, you can write the text output to a file with the File menu Export As command.

You can enable and disable a header section for the entire output and a summary section of all the channel information. You can also set the number of columns to be used when writing waveform and event times and the separators used between fields and to delimit text. For each type of channel you can enable the channel synopsis and data output. You can also choose to dump a channel type in its native format, or in any compatible format. Output sections are preceded by a keyword so that other programs can parse the file.



Time range

Select the time range of data to copy as text in this dialog. Either type in the range, or use the drop down lists to select start and end times for the data output. When you are satisfied with the range, click on **Cancel** or **OK** to start the output.

Spreadsheet format

It is sometimes more convenient to export data as text in a spreadsheet format. This is often simpler to interpret and import into other programs.

Format

All time view Copy As Text information is written in fields. Each field is either numeric or text. Between each field there is a separator, which can be set to be a Tab character, a blank or a comma in the **Separator** drop down list. Most programs that accept tabulated numbers will accept space, a Tab or a comma. The examples below use space as a separator.

A numeric field holds numbers only, either floating point numbers with a decimal point, or integer numbers. A text field is a sequence of characters that may include spaces. Text fields may hold numbers, but numeric fields cannot hold text. You can mark the start and end of a text field with a special character (usually ") so that a program reading the field can include blanks (spaces) and punctuation within a field without confusion. All keywords are treated as text fields. You can set a one character delimiter in the **String delimiter** field, or leave it blank for none. The examples use " as a delimiter.

Columns

You can choose to have the data for waveform channels and event times written in multiple columns. This does not write one channel per column (use Copy Spreadsheet for that format).

Header

The first part of the output is a header that displays information about the data file. This is given after the keyword `INFORMATION`. The header consists of the file name followed by five lines of the file comment and a blank line. The header block is always 8 lines long (if present); blank file comment lines are not skipped.

```
" INFORMATION"
"demo.smr"
"These five lines"
"contain"
"the file comment"
"for this"
```

```
"data"
```

Summary

The summary section starts with the keyword `SUMMARY` followed by a summary of the channel information for each selected channel. The information given for the channel varies with the channel type. The first three fields are the same for all channels, being the channel number, the channel type and the channel title. The remaining fields are:

Waveform	Units, Ideal rate, Actual rate, Scale, Offset
WaveMark	Units, Ideal rate, Actual rate, Scale, Offset, Points, PreTrig
Event	Predicted mean rate
Marker	No other information

Here is some typical output. The `SUMMARY` section is terminated by a blank line.

```
"SUMMARY"
"1" "Waveform" "Sinusoid" "Volts" 100.0 100.0 1.0 0.0
"2" "Evt-" "Synch" 5.0
"4" "WaveMark" "Shapes" "Volts" 100.0 100.0 1.0 0.0 30 5
"31" "Marker" "untitled"
```

Channel information

Time view Copy As Text output format

The Channel section of the dialog lists the channels that are to be output. You can control the output based on the type of each channel. For each channel selected, the output starts with a channel information section that is introduced by the keyword `CHANNEL`, followed by the channel number. The channel number is a text field. If the `Synopsis` box that matches the channel type is checked, the channel data type, comment and title are output on the next three lines. The channel information block ends with a blank line.

```
"CHANNEL" "1"
"Waveform"
"Signal generator"
"Sinusoid"
```

The information that follows the channel information varies with the channel type and is only output if the `Data` field is checked for the channel type. If a channel type is derived from a Marker, you can choose to output the data as though the channel were a simpler type to reduce the quantity of information. For example, if you were only interested in the times of WaveMark data, you can copy it as though it were Event data.

Waveform

A channel containing waveform data has the channel information followed by the data. The data starts with the channel units and ideal sample rate. The next line contains the keyword `START` followed by the start time of the data in seconds and the time increment per data point, also in seconds. It is possible for gaps to occur in the waveform data. A gap is shown by a line with the word `GAP` followed by the start time of the new section and the time increment. The data then follows as a list of waveform values.

```
"CHANNEL" "1"
"Waveform"
"Signal generator"
"Sinusoid"
"Volts" 100.0000
"START" 0.00 0.01
-3.7208
-3.8356
"GAP" 4.23 0.01
-2.2901
-2.6075
-2.8858
```

Event

Event data comes in three types, rising edge, falling edge and both edge triggered (`Evt-`, `Evt+` and `Level`). These three types are represented in basically the same way.

```
"CHANNEL" "2"
```

```
"Evt-"
"Synchronisation pulse from Signal generator"
"Synch"
0.2022
.
.
```

In the case of an event channel triggered on both edges, the state before the first transition is shown at the start of the data, by one of the words HIGH or LOW. HIGH means that the first time in the list represents a low to high transition, LOW means the reverse. If there are no times in the list, these labels are the opposite of the state in the period.

```
"HIGH"
0.2022
.
.
```

Marker

Marker data consists of a series of times and 4 bytes of marker information. The channel synopsis is as for other channels, and the data is displayed as a time followed by a string of 4 characters which represent the 4 bytes of information. If the bytes are non-printing characters a ? is shown instead. The 4 bytes are also displayed as numbers after the string. You can force marker data to be dumped as event data.

```
"CHANNEL" "16"
"Marker"
"No comment"
"untitled"
1.0906 "p???" 112 0 0 0
3.0336 "a???" 97 0 0 0
5.9802 "u???" 117 0 0 0
6.9018 "l???" 108 0 0 0
```

WaveMark

WaveMark data is structured as a marker plus a short section of waveform data. The channel synopsis is the same as for waveform data, but also has the number of waveform points associated with each event and the number of pre-trigger points. The dump of data starts with a line holding the channel units, the ideal sampling rate, the number of data points and the number of pre-trigger points:

Units, Ideal rate, Points, PreTrig

This is followed by blocks of data, one for each WaveMark event in the time range. The first line of the block holds the keyword WAVMK, followed by the time of the first data point in the event, the time interval between waveform points and the four marker bytes:

WAVMK,eventTime,adcInt,mark0,mark1,mark2,mark3

The following lines in the data block hold the waveform data values. This block is repeated for each event in the time range. A typical WaveMark channel begins:

```
"CHANNEL" "1"
"WaveMark"
"Channel comment for demonstration"
"Title"
"Volts" 100.0 30 5
WAVMK 0.0210 0.01 1 0 0 0
0.1318
.
.
"WAVMK" 0.8110 0.01 3 0 0 0
.
.
```

You can also force a WaveMark channel to be dumped as though it was a marker channel, as an event channel or as a waveform channel.

RealMark

The first line of RealMark data output holds the channel units followed by the number of reals attached to each event. This is followed by data blocks flagged by REALMARK followed by the event time and the four marker bytes. The following lines in the data block hold the real number. You can force RealMark data to be dumped as Marker or event data. Typical data follows:

```
"Units" 2
"REALMARK" 1.0906 112 0 0 0
1.8923
4.8567
"REALMARK" 1.8603 113 0 0 0
1.8224
4.123
```

TextMark

The first line holds the maximum number of characters allowed. This is followed by data blocks separated by blank lines, one for each TextMark. The first line of each block starts with `TEXTMARK`, followed by the marker time and the four marker bytes. The second holds the text. You can force TextMark data to be dumped as Marker or Event data.

```
100
"TEXTMARK" 0.5234 112 0 0 0
"This is where I added the secret ingredient"
"TEXTMARK" 9.8603 113 0 0 0
"Control point 1"
```

You can also export this data type by double clicking any TextMark and copying from the TextMark dialog.

Paste

You can paste the text on the clipboard into a text, script or output sequence document. When you paste text, Spike2 checks that each pasted line ends with the correct end of line characters (for example, Macintosh and Windows use different sequences). The paste operation corrects incorrect sequences. To correct text that has come from a different operating system and that looks strange, select all the text, cut, then paste.

You can visualise the end of line characters in a Spike2 text window by opening the appropriate text settings window with the View menu **Font** command and check **Line ends**.

Delete

This command is used to delete the current text selection, or if there is no selection, it deletes the character to the right of the text caret. Do not confuse this with **Clear**, which in a text field is the equivalent of **Select All** followed by **Delete**.

Clear

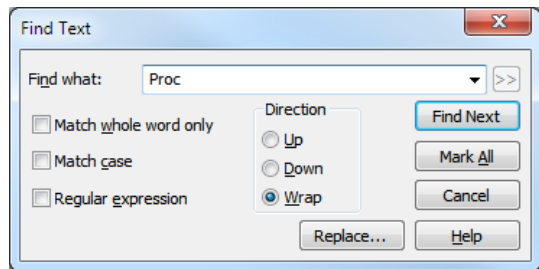
When you are working with editable text, this command will delete it all. If you are in a result window, this command will set all the bins to zero and redraw the window contents. **Clear** removes everything; **Delete** removes the current selection.

Select All

This command is available in all text-based windows and selects all the text, usually in preparation for a copy to the Clipboard command. The short-cut keyboard command is `Ctrl+A`.

Find, Find Again, Find Last

The Edit menu Find command opens the Find Text dialog. The dialog is shared between all text-based views. It is closely linked to the Find and Replace Text dialog and shares all its fields with it; opening this dialog closes the Find and Replace dialog. Click **Replace...** to swap to the Find and Replace dialog.



A successful search moves the text selection to the next matching string. Searches are done line by line (you cannot search for text that spans more than one line). Find Next starts a search for the text in the Find what field. The Mark All button sets a bookmark on all lines that match the search text, but does not move the selection. Searches are insensitive to the case of characters, unless you check Match Case. Check the Match whole word only box to restrict the search so that the first search character must be the start of a word and the last search character must be the end of a word.

The script language equivalent of this dialog is the `EditFind()` command.

Search direction

Select **Up** to search backwards, towards the start of the text. Select **Down** to search forwards towards the end of the text. Select **Wrap** to search forwards to the end, then wrap around to the start and stop when you reach the current position. Searches do not include the currently selected text.

Regular expression

Check this box to search for *regular expressions*. This disables Match whole word only as regular expressions have their own way to match word starts and word ends. It also disables Up searches; regular expression searches are forwards only. It enables the >> button, which displays a list of regular expressions to insert into the search string. The simplest pattern matching characters are:

- ^ Start-of-line marker. This character must be at the start of the search text. The following search text will only be matched if it is found at the start of a line.
- \$ End-of-line marker. This must occur at the end of the search text. The preceding text will only be matched if it is found at the end of a line.
- .
- Matches any character.

To treat these special characters as normal characters with regular expressions enabled, put a backslash before them. A search for “`^\^.\.`” would find all lines with a “`^`” as the first character, anything as a second character and a period as the third character.

You can use `\a`, `\b`, `\f`, `\n`, `\r`, `\t`, `\v` to match the ASCII characters BELL, BS (backspace), FF (form feed), LF (line feed), CR (carriage return), TAB and VT (vertical tab). Searching for `\n` and `\r` will not normally match anything as `\n` or `\r\n` mark line ends and the search is of complete lines ignoring end of line markers.

To search for one of a list of alternative characters, enclose the list in square brackets, for example `[aeiou]` will find any vowel. For a character range use a hyphen to link the start and end of the range. For example, `[a-zA-Z0-9]` matches any alphanumeric character. To include the `-` character in a search, place it first or last. To include `]` in the list, place it first. To search for any character that is not in a list, place a `^` as the first character. For example, `[^aeiou]` finds any non-vowel character.

There are search characters that control how many times to find a particular character. These characters follow the character to search for:

- * Match 0 or more of the previous character. So `51*2` matches `52`, `512`, `5112`, `51112` and `h`. `*1` matches `h1`, `he1`, `hail` and `B[aeiou]*r` matches `Br`, `Bear` and `Beer`.
- + Match 1 or more. The same as “*”, but there must be at least one matching character.
- ? Match 0 or 1 of previous character. So `51?2` matches `512` and `52`; `Bee?t` matches `Bet` and `Beet`.

Put a backslash before these characters to treat them as normal characters in a regular expression.

You can search for the start and end of a word with `\<` and `\>`. Word characters are the set `[a-zA-Z0-9]`, or `[a-zA-Z0-9%$]` in script views. For example, the regular expression `\<[a-zA-Z]+>` will match a text word, but not if it contains numbers. You can also use `\w` to match a word character and `\W` to match not a

non-word character. Likewise, `\d` matches a decimal number and `\D` to matches a non-number character and `\s` matches white space (space, TAB, FF, LF and VT) and `\S` matches non-white space. You can use `\w`, `\W`, `\d`, `\D`, `\s` and `\S` both inside and outside square brackets.

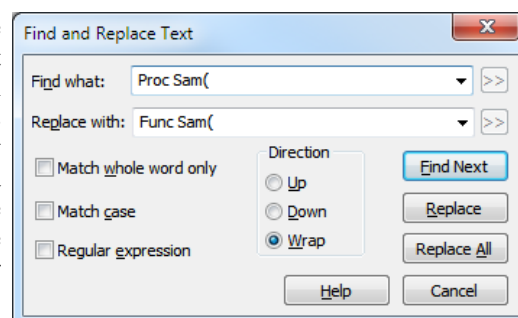
You can match a character with a particular code using `\xnn`, where `nn` is the character code in hexadecimal.

You can also tag sections of matched text by wrapping it in `\(` and `\)`. You can then insert the tagged text later in the regular expression (or in the replace text in the Find and Replace dialog) using `\n`, where `n` is 1 for the first remembered text, up to 9 for the ninth. For example, `\(foo\)-\1` matches `foo-foo`. More interestingly, the regular expression `\(\<[a-zA-Z]+\>\)-\1` matches `Jim-Jim`, `plum-plum` and the like.

Find Again and Find Last repeat the current search forwards or backwards.

Replace

The Edit menu **Replace** command is available when the current view is text-based. It opens the Find and Replace Text dialog in which you can search for text matching a pattern and optionally replace it. The search part of the dialog is identical to the Find Text dialog. The search pattern set by the Find what field can be a simple match, or can be a regular expression. In regular expression searches, the replacement text can refer back to tagged matches in the search text. See the Find Dialog for details of regular expressions and tagged matches. The `>>` buttons are also enabled in regular expression mode and let you insert expressions into the search and replace text.



The script language equivalent of this dialog is the `EditReplace()` command.

Replace with

This field holds the text to replace the matched search text. In a regular expression search, you can include tagged matches from the search text using `\1` to `\9` as described for the Find dialog. For example, suppose you have variables named `fred0` to `fred17` that you want to convert into an array `fred[0]` to `fred[17]`. You can do this by setting the Find what field to `\<fred\([0-9]+\)\>` and the Replace with field to `fred[\1]`.

Replace

The **Replace** button checks that the current selection matches the Find what field and if it does, the field is replaced by the Replace with text field and the selection is moved to the next match. If the current selection does not match, **Replace** is equivalent to **Find Next**.

Replace All

This button searches for all matches in a forward direction starting from the beginning of the text to the end, and replaces them.

Edit toolbar

The Edit toolbar gives you access to the edit window bookmarks and short cuts to the Find, Find next, Find previous and Replace commands. If you are unsure of the action of a button, move the mouse pointer over the button and leave it for a second or so; a “Tooltip” will reveal the button function and any short-cut key associated with it.

The four bookmark functions toggle a bookmark on the current line, go to the next or previous bookmark and clear all bookmarks. The Edit menu Find command can set bookmarks on all lines containing a search string.

If the Edit toolbar is not visible, right-click on an empty area of a toolbar or of the Spike2 main window. Then select **Edit bar** in the pop-up menu. You can use the same procedure to hide it. The bar can be docked to any side of the application main window or dragged off the application main window as a floating bar.

Auto Format

Automatic formatting is available for script views and applies standard formatting while you type and lets you reformat entire scripts or selected regions. Formatting is done by indenting lines of text based on the script keywords. An indented line is one that starts with white space. The indenting is in units of the Tab size set for the view in the **Script Editor Settings** dialog. Indentation is done with Tab characters if you have chosen to keep Tabs in the view, otherwise indentation is done with space characters. There are two sub-commands:

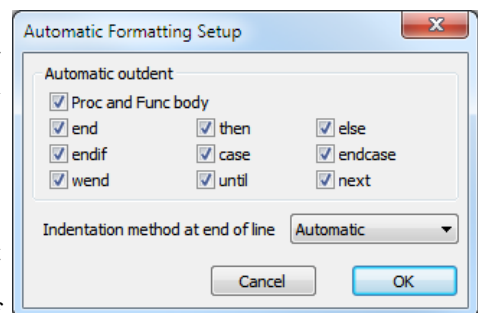
Apply Formatting

If any text is selected in the current view, all lines included in the selection are formatted. If there is no selection, the entire document is formatted. If text is selected, you can right-click in the text view and choose **Auto Format Selection** from the pop-up menu to activate this option.

Settings...

The **Automatic Formatting Setup** dialog controls how text is formatted. Formatting is based on the same scheme that is used for folding text. Each script keyword that starts a block construction (`proc`, `func`, `if`, `docase`, `while`, `repeat`, `for`) increases the indent, and the keywords that complete blocks decrease the indent.

The standard CED formatting is to have all the boxes checked, however, it makes no difference to the script operation so, what you choose is a matter of personal taste. It is a good idea to use consistent indenting as it helps you to understand the structure of the script.



Proc and Func body

If this box is not checked, all text within a function or procedure is indented. Check the box to outdent the text between the `Proc` or `Func` and the `end`.

end...next

You can choose to outdent any line starting with one of the keywords `end`, `then`, `else`, `endif`, `case`, `endcase`, `wend`, `until` and `next`.

Indentation method at end of line

This field determines what happens to the indentation of the current line and the new line when you type the `Enter` key. The settings are:

None No automatic formatting is applied. The new line is not indented.

Maintain The new line is indented to match the indentation of the current line.

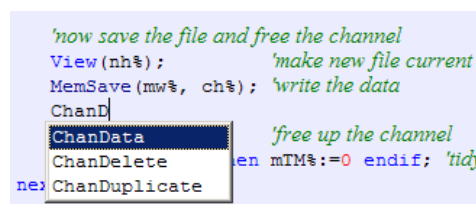
Automatic The indentation of both current and new lines is adjusted based on the Auto Format settings.

Toggle Comments

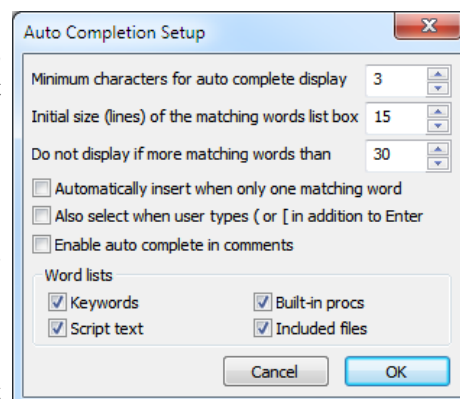
This Edit menu command is available in script and text sequencer views, and is also available in the context menu when there is a selection. It adds or removes a comment marker at the start of the line for all lines in the current selection. It decides what to based on the first character of the first line in the selection.

Auto Complete

In any script, you will find that you are typing the same text items repeatedly. The editor can save you some time by popping up lists of known words that match your typing. The matching is done by looking for a word break in the text before the text caret, then matching your typing against various categories of words known to the script. Matched words are displayed in alphabetical order and the current word is highlighted.



You can use the up and down arrow keys to choose an item in the list and the `Enter` key to select an item or double click an item to enter it. The `Esc` key cancels the list. Alternatively, you can just keep on typing, which will narrow the choice of words to match. The Edit menu **Auto Complete Setup** dialog gives you some control over the words that are matched and when the auto-completion lists appears.



The **Word lists** section of the dialog lets you choose the categories of words to match your typing against. If you do not want to display auto-completion lists, clear all the categories.

You can choose from script keywords (`Proc`, `Func`, `EndCase`, ...), built-in functions and procedures (`App`, `NextTime`, ...), words that already exist in the script and user-defined `Func` and `Proc` names in included files. You may wish to disable the **Script text** option if you are working with a huge file and you notice an appreciable delay after typing before the list appears.

Minimum characters for auto complete display

This sets the number of characters in a name that must be typed before the list will appear. You can set this to 1 to 12 characters. Setting a low value can cause the pop-up display to become annoying. You need to choose a count that gives you enough help, but not too much. Try a value of 3 to start with.

Initial size (lines) of the matching words list box

This field sets the maximum number of words to display at a time in the range 1 to 40. If there are more matching words, the list contains a scroll bar. After the list appears, you can re-size it by clicking and dragging on the horizontal edge furthest away from the matched text.

Do not display if more matching words than

If there are more matching words than you set here, the list is not displayed. You can set this value in the range 1 to 200 words.

Automatically insert when only one matching word

Because of the design of the script language, apart from variable declarations, all words you type in are likely to have been defined already. If you set this option, once your typing has reduced the list size to 1, the word is automatically inserted. You may want to increase the **Minimum characters for auto complete display** if you enable this option.

Also select when user types (or [

Normally, the list text is inserted when you type the `Enter` key or double-click the list text. If you check this option, the text is also inserted when you type an opening round or square bracket, followed by the bracket.

Enable auto complete in comments

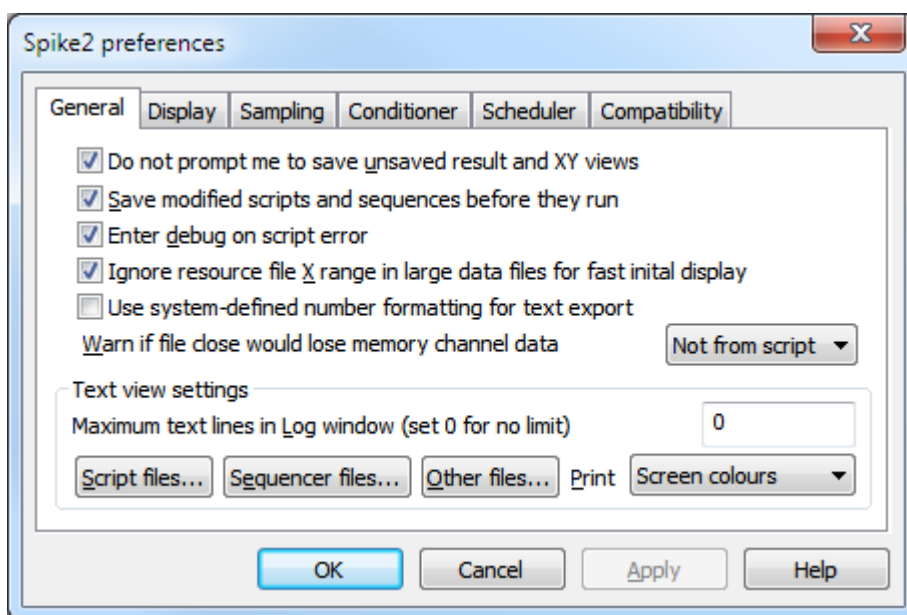
Normally, automatic word completion is disabled in a comment. However, if you need to refer to script functions in your code documentation you may find it useful to check this box.

Preferences

The Edit menu Preferences dialog has tabs for general settings, display and sampling options, signal conditioner, time scheduling and for compatibility with previous Spike2 versions. Preferences are stored in the Windows registry and are user specific. If you have several different logons set for your computer, each has its own preferences. You can use the `Profile()` command to change the preferences from a script.

General

This tab holds editor preferences and general settings for saving modified views and to control what happens when an error is found in a running script.



Do not prompt me to save unsaved result and XY views

As it is often possible to recreate result and XY from the raw data, you can check this box to suppress the normal Windows behaviour of prompting you to save unsaved data.

Save modified scripts and sequences before they run

If you check this box and run a modified script or sequence, it will be saved first. If the script has no name, the File Save dialog opens to prompt you for a file name.

Enter debug on script error

Normally, if a running script has a problem, it stops with an error message in the script window and Log view. If you check this box, the script debugger is activated on a script error and you can inspect the local and global errors and the call stack at the time of the error. You are not allowed to continue running the script.

Ignore resource file X range in large data files for fast initial display

Normally, when Spike2 opens a data file with an associated resource file, it shows the same x range as was visible when the file closed. If the file is very large this initial display can take several seconds. If you check this option, when you open a data file that is more than 50 MB in size, Spike2 displays the first second of data and not the range displayed when the file closed.

Use system defined number formatting for text export

Normally, Spike2 uses standard compatibility options for all text output. If you check this option the number formatting settings defined in the operating system are used for data exported to the clipboard or file as text. The main effect is that if your operating system is set to use a comma character as the decimal point, you can

now export numbers using this. This has no effect on data input and no effect on the script language.

Warn if file close would lose memory channel data

Memory buffer channels only exist while the data file is open and are lost when you close the file. In many cases you will not care about this, however, if you have put considerable effort into creating a memory buffer you may wish to be warned that you are about to lose one. You are not warned if there are memory channels, but they contain no data. This feature was added at Spike2 version 8.02. There are three warning levels:

- | | |
|-----------------|--|
| Never | You will never be warned. This was the state before Spike2 version 8.02. |
| Not from script | You are warned unless the file is closed with the script <code>FileClose()</code> command. This is the default warning level. It is chosen as the level that will not cause a any script written before version 8.02 to stop with a warning, but it will warn of any interactive use that might lose data. |
| Always | You are always warned unless the file is closed with the script <code>FileClose()</code> command and <code>query%</code> is set to -1. |

Text view settings

These controls configure text-based views (script, sequencer, log and script-created text views) on screen and when printing. The dialogs accessed by the **Script**, **Sequencer** and **Other** files buttons apply changes globally (to all open files of the selected type and to all future files). You can use the same dialogs from the **View** menu **Font** commands to change the current view (there is a button to apply changes to all views).

Maximum text lines in Log window

This field prevents huge amounts of text accumulating in the Log view. If there are more lines than set here in the Log view and a script writes more, the oldest lines are deleted to leave this number. To avoid deleting lines every time a line is added (which is very slow), each time the line count exceeds the number of lines set here, the excess lines plus 10% of the maximum lines are deleted so that the view can grow normally for a while. Set 0 for no line number limit. Lines are not deleted as a result of interactive typing in the window. The script language equivalent is the `ViewMaxLines()` command.

Print

This sets how to print coloured text from a text view. The choices are:

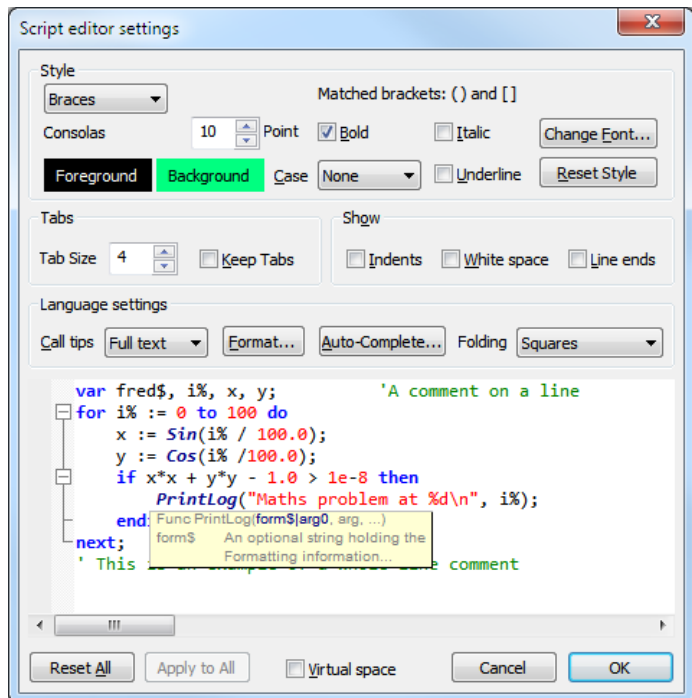
- | | |
|-----------------|--|
| Screen colours | Use the displayed screen colours; this uses a lot of ink if the background is not white. |
| Invert light | If you display your text as a light colour on a dark background, this setting prints on a white background and inverts the text colours. |
| Black on White | Prints black text on a white background. |
| Colour on White | Prints in screen colours on a white background. |

Script file text settings

Open the editor settings dialog from **Edit** menu **Preferences General** tab to change settings for all views of the current type, or from the **View** menu **Font** command for the current view; you can use **Apply to All** to apply changes to all views. The **Reset All** button returns all values to standard settings. The bottom half of the dialog holds example text to preview any changes.

Virtual space

Check this box, to enable virtual space in the editor. This allows you to move the text caret beyond the end of a line by clicking with the mouse or by using cursor keys. This allows you to position text (for example comments) without the need to use tabs or spaces to position the caret. If you type with the caret in virtual space, the editor inserts spaces up to the caret position and the position becomes "real". However, allowing virtual space may force you to use extra key presses to move to the real start or end of a line (**Home/End** keys) when you press cursor up and down to move vertically through the text.



Style

Each view type supports one or more text styles. For each style you can set a font (including proportionally spaced fonts). The font includes the size in points (in the range 2-256) and bold and italic settings. You can also choose to force upper or lower case and underlines for all text in the style. You can set a foreground and background colour for the style by clicking on the **Foreground** and **Background** rectangles.

All views have a **Default** style that the other styles are based on; it is used for everything that is not covered by one of the other styles including the text caret colour. The **Tab** size is based on the width of a space character in this style. If you change any aspect of this style, all the other styles that match that aspect will also change.

In a **Script** view, you can control the appearance of many different aspects of the display, based on the syntax of the language. There are settings for:

- White space** Style used when drawing spaces, tabs and control characters (normally the same as **Default**).
- Comment** The style used when displaying comments.
- Numbers** The style to use when displaying numbers.
- Keywords** The style to use for script keywords, such as `for`, `next` and `end`.
- String** The style for literal strings, such as `"This is a string"`.
- Function** Used for built-in script functions, such as `PrintLog()`.
- Operator** The style for script language operators, such as `+`, `-` and `+=`.
- Identifier** The style for function, procedure and variable names created in a script.
- Line numbers** The style used for line numbers and the gutter. When you select this item, line numbers will appear in the example text area.
- Call tips** The style for pop-up call tips. This style has an extra **Tip highlight** colour field, used to highlight the current argument in a function. This replaces the **Bold**, **Italic**, **Case** and **Underline** fields, which are not displayed.
- Braces** The style for matched square and round brackets `()` and `[]`. When the text cursor is next to a bracket, the editor searches for the matching bracket and applies this style if it is located.
- Bad braces** The style to use when the text cursor is next to an unmatched bracket.

Reset Style and Reset All

The **Reset Style** button reverts the current style to standard settings. **Reset All** reverts all styles.

Tabs

This dialog region controls the size of tabs (set in units of the width of a space character in the **Default** style) and if Tabs are implemented by saving a Tab character in the text (check **Keep Tabs**) or are implemented as multiple spaces (clear the **Keep Tabs** check box). If you use a fixed pitch font, such as *Courier New*, then it does not matter too much if you choose to keep Tab characters or not. If you use a proportional font for anything except comments, it is better to keep the Tab characters. When automatically formatting a script, the **Keep Tabs** setting determines if indents are generated with Tab characters or spaces.

Show

In addition to displaying the text, you can also choose to display information about the white space in your file. The settings are:

- Indents** It can be useful when manually lining up indented loops in a script to see the indent level. Check this box to display vertical lines at each tab stop in the leading white space of each line.
- White space** Check this box to display spaces with a centred dot and tabs as right arrows.
- Line ends** Check this box to see the Carriage Return (CR) and Line Feed (LF) characters that mark the end of a line. This is useful when working with scripts that move the caret around.

Language settings

This area of the dialog is used by script and output sequencer views. It sets your preferences for call tips, automatic formatting, automatic text completion and folding. The **Format** and **Auto Complete** buttons duplicate Edit menu commands, and are described separately.

Folding

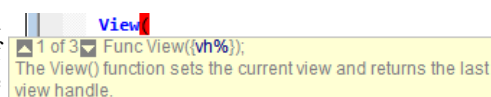
Script views and output sequence views can display a folding margin, and allow you to fold the code by clicking in the margin, from the **View** menu **Folding** command, and by right clicking in the view and selecting **Toggle all folds**. In a script view, fold points are based on a lexical analysis of the script text. You can choose from one of four folding styles, or have no folding margin.

Call tips

A Call tip is a block of text that pops up in a script view when you type the opening brace of a function or procedure name or you hover the mouse pointer over a procedure or function name. You can choose from **None** (no tips), **Single line** (the command name and arguments) and **Full text** (includes a command synopsis).

Call tip details

The call tip text contains the command name and arguments and a synopsis of the command. If the tip appears as a result of typing the opening brace, the editor attempts to highlight the current argument in the tip and if the command has multiple variants, you can select the variant to display by clicking on the arrows in the call tip. In **Single line** mode (not when opened by hovering the mouse pointer) you can click on the body of the tip to show the full text.



For built-in functions, the call tip text holds the command name and arguments plus one sentence of description. This information comes from the text file `script.tip` in the folder where Spike2 is installed.

For user-defined `PROC` and `FUNC` items, the arguments are taken from the line containing the `PROC` or `FUNC` keyword (expected to be the first item on the line). If the line before the `PROC` or `FUNC` is a comment, up to 10 lines of comment are used as a description. This works for `PROC` and `FUNC` items in the current source and in any included files. If the line before is blank, and the line after is a comment, up to 10 lines following are used as a description. Putting the comments after the `PROC` or `FUNC` line has the advantage that the comments can be folded within the `PROC` or `FUNC`.

The search is terminated by any "divider" lines. A divider line is one that is at least 10 characters long (after

removing the initial comment mark ') and with no more than 2 characters being different. Here are some example divider lines:

```
.....
'|-----|
'|=====|
```

If you document your functions and procedures as in this example, they will generate tidy call tips:

```
'This is an example of how to document for a nice call tip
'arg1 If the first word on a line is followed by more than one
'      space or a tab, the rest of the line is indented. If a line
'      starts with two or more spaces or a tab, it is indented.
'arg2 Description of second argument
'arg3 And something about the third argument
Func Example(arg1, arg2, arg3)
var x,y; 'the start of the code
...
```

The example text above would produce a call tip looking like this. The comment markers at the start of each line are removed, and the multiple spaces after the first word or at the start of each line are replaced with a Tab character.

```
Example (
Func Example(arg1, arg2, arg3)
This is an example of how to document for a nice call tip
arg1 If the first word on a line is followed by more than one
      space or a tab, the rest of the line is indented. If a line
      starts with two or more spaces or a tab, it is indented.
arg2 Description of second argument
arg3 And something about the third argument
```

If you do not want a Proc or Func to have a call tip or to appear in an Auto-completion list, put the Proc or Func on a separate line from the name. This can be useful when you build a library of functions to be placed in an include file and you only want some of the functions to be visible for call tips and Auto-complete. For example:

```
Proc
PrivateFunction() 'This is only used within this file
...
end;
```

Sequencer file text settings

The editor settings dialog for output sequencer files is very similar to that for script files. The example text in the lower half of the dialog displays some typical sequencer code, and there is no support for Call tips, auto formatting or automatic completion.

There is a list of styles that you can apply to the different elements of the sequencer text. The Default style is used in exactly the same way as for script views as the basis for all other styles and as the font that is measured to set the Tab size. The remaining styles are:

- White space** Style used when drawing spaces, tabs and control characters (normally the same as Default).
- Comment** The style used when displaying comments.
- Numbers** The style to use when displaying numbers.
- Keywords** The style to use for standard output sequencer instructions.
- Deprecated** The style to use for outmoded output sequencer instructions like DAC0 that have been replaced and may not be supported in future revisions.
- Display** Text introduced by > for display on screen during sampling.
- Operator** The style for mathematical operators like +, -, * and /.
- Identifier** The style for user-defined variable names and labels.
- Functions** The style for built-in conversion functions like ms().
- Step** The style for the optional step numbers at the start of an instruction line.
- Key** The style for keyboard links introduced by a single quote, for example 'H
- Line numbers** The style used for line numbers and the gutter. When you select this item, line numbers will appear in the example text area.
- Directives** Items such as SET and VAR that do not generate output instructions.
- Braces** The style for matched square and round brackets () and []. When the text cursor is next to a bracket, the editor searches for the matching bracket and applies this style if it is located.

Bad braces The style to use when the text cursor is next to an unmatched bracket.

Folding support

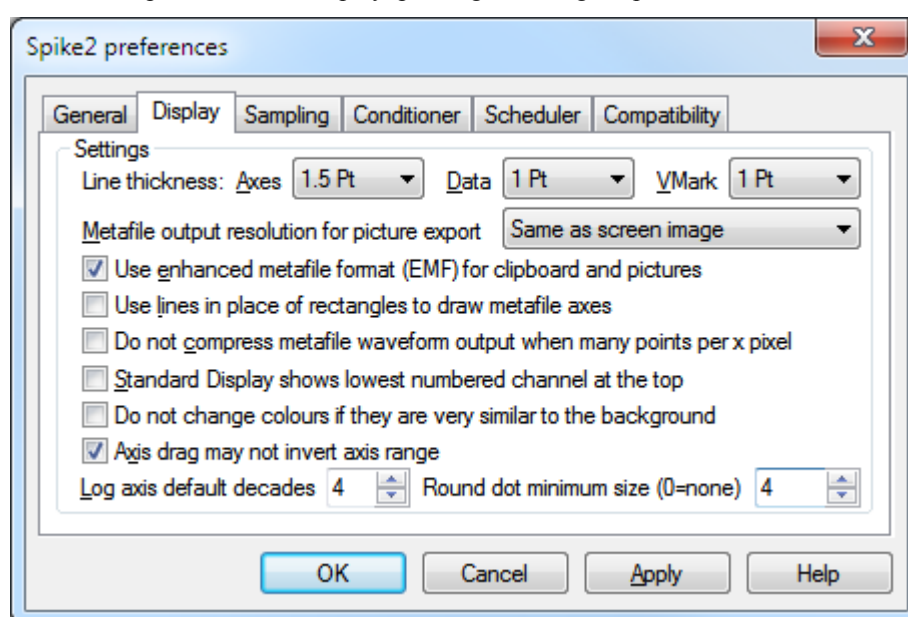
The output sequencer editor supports code folding based on the keyboard link entry points. That is, you can fold up all code from a line holding a keyboard link up to the next line holding a link.

Other text file settings

The editor settings dialog for all views except script and sequencer views is the same as the dialog for script views, except that there are no language settings and there is only the Default and the Line number text style.

Display

This dialog tab contains options for data display, printing and image export.



Line thickness: Axes, Data and VMark (Vertical Markers)

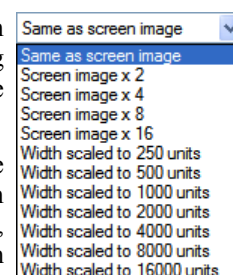
Choose from Hairline (as thin as possible) and a list of point sizes. A point is 1/72nd of an inch, which is typically about the size of a screen pixel. If coloured lines do not print on a monochrome printer, increase the line thickness or use the View menu Use Black and White command. Click Apply to redraw screen images to show the effect of any change. WARNING: wide lines take longer to draw than a single pixel line.

You can override line thicknesses for particular channels with the Pen Width and XY Draw Mode dialogs. Channels drawn as dots set the dot size as a multiple of the pen width for data set in this dialog; so if you set a thick pen here, you will get large dots on screen.

Metafile output resolution

Spike2 saves time, result or XY views as pictures in either bitmap (a copy of the screen image) or metafile format. A metafile describes an image in terms of drawing operations. You can set the metafile drawing resolution; the higher the resolution, the more detail in the picture.

A problem for time and result views with many data points is that the higher the resolution, the more lines need to be drawn, and many drawing programs have limits on the number of lines they can cope with. You can use multiples of the screen resolution, or fixed resolutions. If you are not sure what setting to use, start with Same as screen image and adjust it as seems appropriate.



Use enhanced Metafile format

Spike2 supports two metafile formats: Windows Metafile (WMF) and Enhanced Metafile (EMF). WMF is a relic of 16-bit Windows and has limitations, but is widely supported. EMF is the standard for 32-bit programs and has many more features. However, some graphics packages do not support this fully.

With EMF format you can export waveforms in cubic spline mode and sonogram data as part of the metafile. With WMF format, waveforms in cubic spline mode export as plain lines and sonograms will be blank.

Use lines in place of rectangles...

This affects metafile output. Some graphics programs cannot cope with axes drawn as rectangles; check this box to draw axes as lines. We use rectangles to make sure that axes drawn with pens of other than hairline thickness join neatly.

Do not compress metafile waveform output

When Spike2 draws waveform data, it does not waste time with lines that make no visual difference to the output. If there are more than 3 data points per horizontal pixel, Spike2 draws one vertical line per horizontal pixel to give the same visual effect as all the separate lines. However, when you export an image as a metafile this may not look correct as it relies on a perfect match between the vertical line width and separation.

If you check this box, no compression is done when data is saved as a metafile. For the most precise image, set the Metafile output resolution to 16000 units and check this box. When you import a metafile, most drawing programs will preserve the waveform as connected lines. Spike2 writes long sequences of data points in blocks of up to 4000 points. The last point of a block is at the same position as the first point of the next.

Standard display shows the lowest numbered channels at the top

You can change the order of time and result view channels by clicking and dragging channel numbers. This field sets the order when you use the Standard display command or create a new channel. If the box is unchecked, lower numbered channels are at the bottom.

Axis drag may not invert y axis range

If you check this box, a drag of a y axis or the x axis in an XY view to change the scale (a drag starting in the number label region) is not allowed to change the direction of the axis. If you check the box you can still edit the axis limits to invert the sense of the axis by double clicking the axis to open the axis dialog.

Do not change colours if they are very similar to the background

Spike2 checks how similar the colours of items such as channel data and text are to the background. If they are too similar, your colour choice is ignored and a more visible colour is used. Check this box to disable the colour check.

There are two levels of contrast: high-contrast is used when you need to read text or when dealing with very small dots, normal contrast is used for everything else. In high-contrast mode, if the foreground and background are similar in brightness, the foreground is set to white or black, whichever is most different from the background. In low-contrast mode, the foreground colour is modified by adding or subtracting a light grey from the colour (this attempts to preserve some idea of the original colour).

Log axis default decades

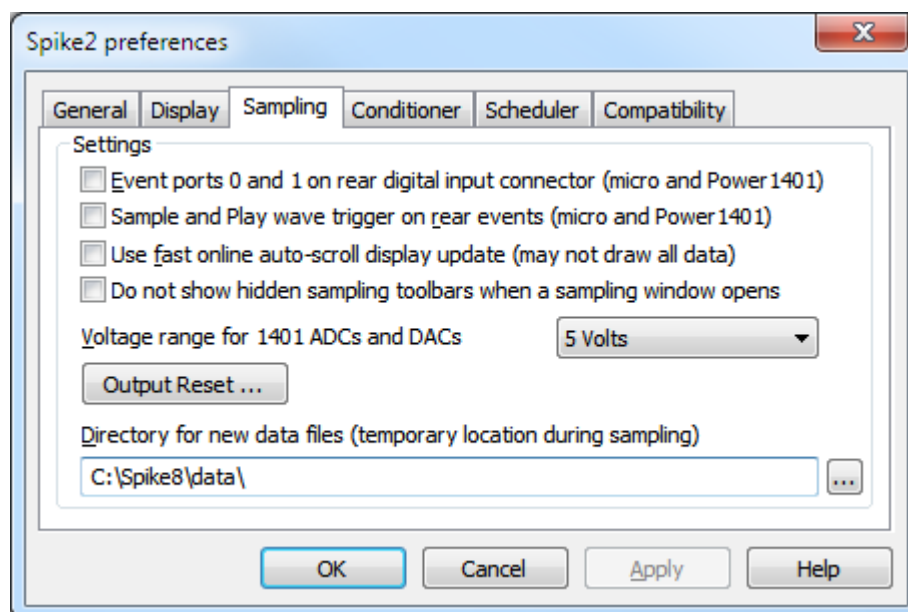
This field sets the number of decades to display when you swap an axis to logarithmic mode from the X or Y axis dialogs or Optimise the Y axis and the low limit is less than or equal to 0. On a logarithmic scale, negative numbers do not exist (as real numbers), and $\log(0)$ is minus infinity. In these cases, the low axis limit is set to be the high axis limit divided by 10 to the power of the number of decades you set here. For example, with 4 decades, if the upper value was set to 400, the lower value would be set to .04 (that is, $400 / \text{Pow}(10, 4)$).

Round dot minimum size

You can choose to draw round rather than square dots in time and result views. This field sets the minimum size (in pixels) at which to do this or you can set the value 0 to always draw dots as squares (to match older versions of the program). Note that printed output will typically have a much smaller pixel size than the screen, so dots that draw as squares on the screen may draw as circles when printed if you enable this option.

Sampling

This tab sets preferences associated with sampling data and the 1401.



Event ports 0 and 1 on rear digital input connector

You can source event ports 0 and 1 from the front panel or the digital input connector. Check the box to use the rear panel. Leave it unchecked to source your event signals from the front panel Event 0 and 1 inputs.

Sample and Play wave trigger on rear events

You can trigger sampling and on-line waveform output with the front panel Trigger BNC, or on the rear panel Events connector pin 4 (GND is pins 9-15). Check the box to use the rear connector.

Use fast online auto-scroll display update

The online automatic scrolling of new data into the display during sampling can make the host computer feel unresponsive, especially in complex display modes at high sampling rates. Check this box to use a faster (but incorrect) algorithm to decide how much of the screen to repaint when the view scrolls during sampling.

In the faster mode, Spike2 does not properly allow for changes in already drawn data that depend on newly sampled data that has just scrolled into the display. This is particularly visible with sonograms and channels with channel processing options such as time shifts.

Do not show hidden sampling toolbars when a sampling window opens

When you open a data file for sampling, Spike2 will normally also show the Sample control toolbar, the Sample Status toolbar and the Output Sequence control bar (if there is an output sequence). If you check this box, these toolbars remain in their current state. You may wish to do this to prevent the toolbars changing the screen arrangement.

Voltage range for 1401 ADC and DACs

Most 1401s have a ± 5 Volt input range, but some have ± 10 Volts. Spike2 detects the range of connected Power1401s and Micro1401s automatically. Choose from 5 Volt, 10 Volt and Last seen hardware. You are warned if Spike2 detects a conflict between user settings and installed hardware. The voltage range affects scaling in the sampling configuration and DAC output values in the output sequencer. It has no effect on scale values in previously sampled data files.

Output Reset...

This button opens a dialog in which you can choose to have the Digital and DAC outputs set to known values

when Spike2 starts, before sampling and after sampling. Values set from the Edit menu Preferences can be overridden by values set in the sampling configuration Automation tab.

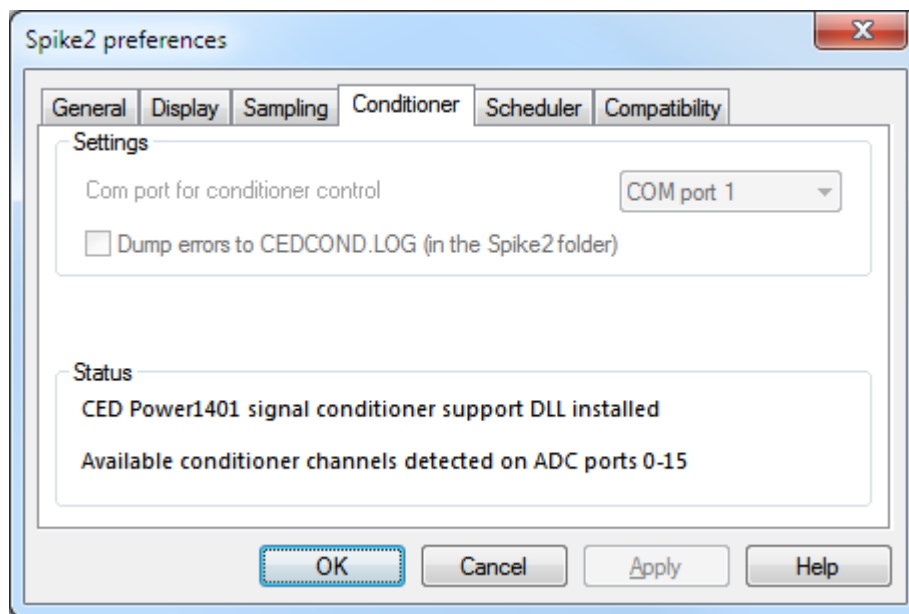
Directory for new data files

This is the directory/folder where Spike2 stores files data created by File menu New during sampling. Click . . . to select a new folder interactively. If you do not set a directory, Spike2 uses the current directory, which may not be where you expect, so it is a better to set one. If you have more than one disk drive, choose a folder in your fastest drive. Do not use a networked drive. New data files in this folder do not have a file extension.

When you close a new data file, Spike2 prompts you for a file name. What happens next depends on where you choose to save the file. If the file is on the same drive as the directory/folder set in the here, Spike2 renames the file (quick). If the drive is not the same, Spike2 copies the file (slow), and deletes the original.

Do not choose a place that requires administrator privilege unless you understand the implications of this.

Conditioner

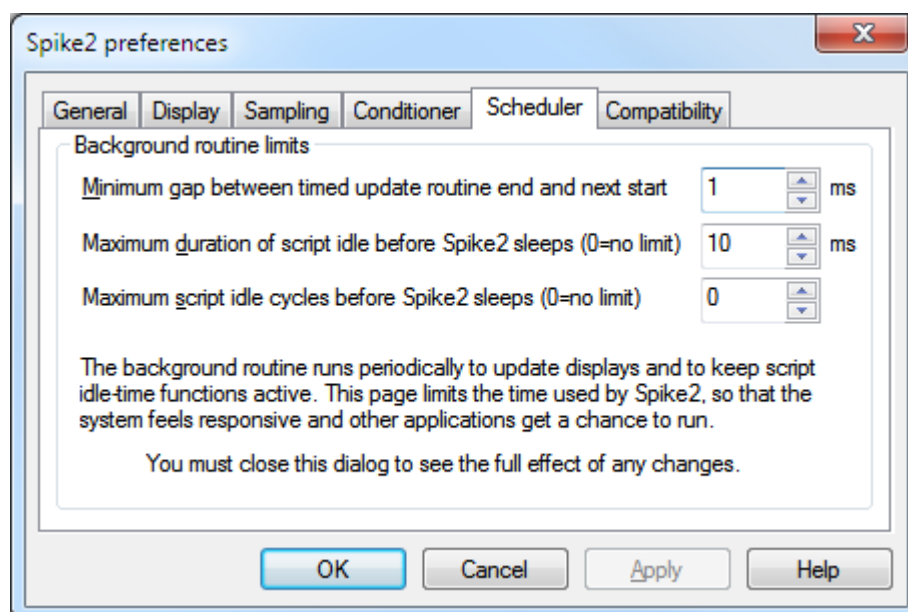


This tab lets you set the port used by a signal conditioner. Check the Dump errors... box to write diagnostic messages to CEDCOND.LOG in the Spike2 folder. If you wish to change the type of conditioner you must run the Spike2 install program; you will be offered a list of conditioners to choose from.

If the currently installed signal conditioner uses a serial line port, you can select any port in the range COM 1 to COM 8. Each time you change the port, Spike2 searches for any conditioner attached to it and will update the status panel.

The Status panel gives information on the type of signal conditioner support that is installed. If it can detect a signal conditioner, it also holds information about the channels that have conditioner support.

Scheduler



This tab limits the processor time consumed by the Spike2 user thread while sampling and idling with a script running. If Spike2 takes too much time, the system feels unresponsive. The tab does not affect the time-critical thread that writes sampled data to disk. You can read more about threads below. Spike2 runs a background routine when it has idle time and also periodically on a timer. The routine handles the following tasks:

- When sampling or rerunning, it gives windows a chance to detect that the maximum time has changed, which may cause windows to scroll and processing to occur. Any invalidated windows will update the next time Spike2 gets idle time.
- If a script is running, it gives any "idle function" in the script a chance to run.
- In automatic file naming mode it starts the next file running.

There are three fields that limit the time used in the background routine. They have no effect on the time used when Spike2 runs a script that does not idle (see the `Yield()` or `YieldSystem()` script commands for this). The standard values work for most cases.

Minimum gap between timed update routine end and next start

If the time interval set by this field passes without the background routine running, it is scheduled to run as soon as possible. You can use values in the range 1 to 200 milliseconds. The standard value is 20 milliseconds. The lower the value, the more time Spike2 will spend on background processing relative to other applications. This field also limits the time that Spike2 will sleep for (see the discussion of threads, below).

Maximum duration of script idle before Spike2 sleeps

When Spike2 gets idle time (see the discussion of threads, below), you can limit the time it uses before Spike2 goes to sleep. Spike2 uses idle time to run script idle routines, such as those created by `ToolbarSet(0, ...)`. You can set from 0 to 200 milliseconds (0 means no time limit). The standard value is 10 milliseconds. The larger the value, the more the script idle routine runs at the expense of other applications.

Maximum script idle cycles before Spike2 sleeps

As an alternative to limiting the script idle routine by elapsed time, you can limit it by the number of times it is called. You can set from 0 to 65535 times (0 means no limit). The standard value is 0. Setting both this and the Maximum duration... field to 0 is unkind to other applications. Setting this to 1 is the most generous to other applications.

Threads

A *thread* is the basic unit of program execution; a thread performs a list of actions, one at a time, in order. To give you the impression that a system with one processor can run multiple tasks simultaneously, the system scheduler hands out time-slices of around 10 milliseconds to the highest priority thread capable of running. Tasks at the same priority level share time-slices on a round robin basis. Lower priority tasks rely on higher priority tasks "going to sleep" when they have nothing to do or when they are blocked (for example, waiting for a disk read). If this did not happen, low priority tasks would not run.

There are also very high priority tasks, usually associated with hardware device drivers, that can interrupt the scheduling system to respond to external events within microseconds. These interrupts usually only last a few microseconds; if a device driver needs a longer period of time it will request a time slice to complete its work and wait for it to be granted.

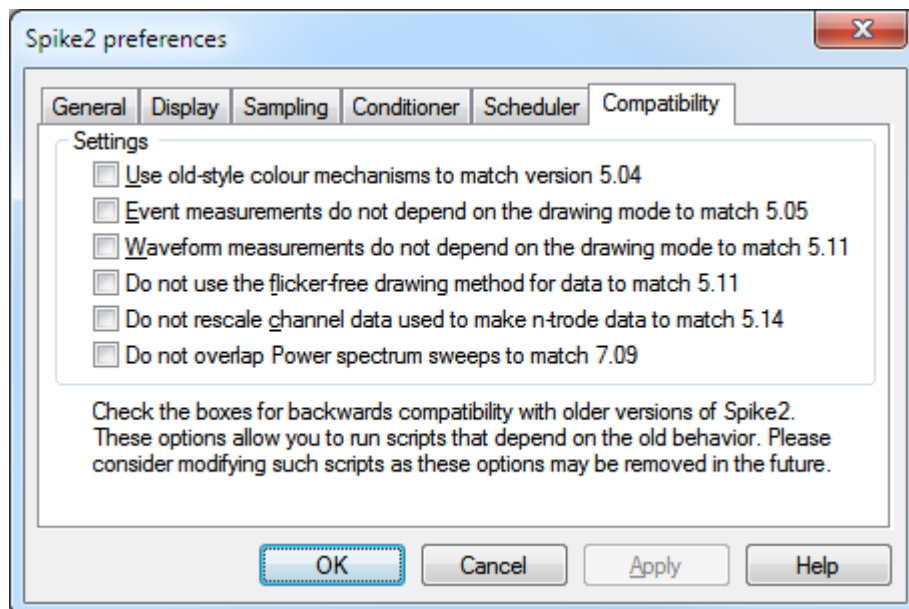
When Spike2 gets a chance to run, it processes pending messages such as button clicks, keyboard commands, mouse actions and timer events and then updates invalid screen areas. Finally, Spike2 is given idle time until it says it does not need any or new messages occur. If Spike2 needs no more idle time it sleeps until a new message appears in the input queue. The `Minimum gap...` timer wakes up Spike2 if nothing else happens.

Most actions in Spike2 (data display, interactive dialogs and the script system) run in the context of a single user thread, so they have to wait for previous actions to end. Separate threads are used for the real-time sampling thread that moves data sampled by the 1401 into the Spike2 data filing system, for the video capture and replay system and for a thread used to calculate FIR filter coefficients in the filtering dialog. These operations can overlap with the user thread actions. In a single processor system, the processor swaps around between all available threads, assigning time to them based on their priority. If your system supports multiple processors or multi-core processors or hyper threading, then threads can run simultaneously, and you will feel the benefit of this in smoother and faster system operation, particularly when sampling data or recording video.

Sampling thread

There is a separate, high-priority thread that is used collect data from the 1401 and channel it into the data file. This makes sure that data is saved, regardless of any user activity.

Compatibility



This tab lets you disable new program features that have changed the way that Spike2 works and that might affect your results. We will keep these items for all version 8 releases; we do not promise to keep them beyond this. If you need to check any of the boxes, please let us know so we have some idea of the number of users who depend on the old behaviour. These settings are saved in the system registry and are read when they are needed.. You can modify the settings with the `Profile()` script command

Use old-style colour mechanisms

Before version 5.04, the colour palette was saved in sampling configuration files; loading a sampling configuration set the colour preferences. We now save the palette in the registry. Check this box to read colours from configuration files, as before (only works for old-style .s2c files).

Event measurements do not depend on the drawing mode

Before version 5.06, measurements taken from event channels drawn as mean frequency or instantaneous frequency returned the count of events between two times. Now, the measured values relate to what you see on screen. Check the box for the old behaviour.

Waveform measurements do not depend on drawing mode

Before version 5.12, measurements from waveform channels returned the nearest data point. Now they return what you see on screen. Check the box for the nearest data point.

Do not use the flicker-free drawing method

Version 5.12 implements a new buffered drawing method that reduces screen flicker on updates. However, this may impact the display speed. Check the box for the old method.

Do not check channel scaling and units in n-trode formation

Version 5.15 onwards scales each channel separately when making n-trode data from multiple waveform channels. Check the box to use the first channel scaling for all channels.

Do not overlap Power spectrum sweeps to match 7.09

From version 7.10, we overlap the Power spectrum sweeps. This will tend to weight the data more equally and can give more consistent results at the cost of more time spent transforming the data. Check this box for the old behaviour.

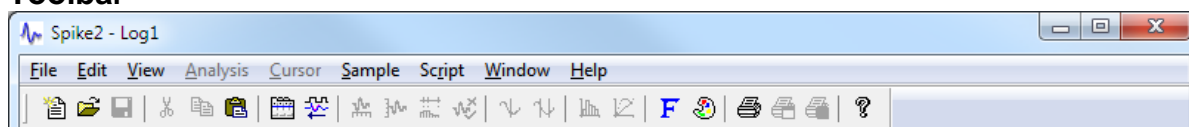
8: View menu

View menu

This menu controls what you see in Spike2 data views and how Spike2 displays the data. Views are generally divided into two basic categories: data-based views (time, result and XY views) and text-based views (script, output sequence, text and log views). The contents of the menu changes depending on the type of the view. The menu also has commands to control the Toolbar and the Status bar.

Toolbar and Status bar

Toolbar



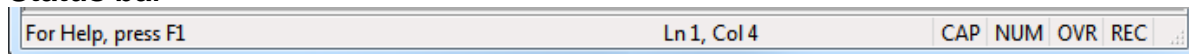
The system toolbar is normally docked below the Spike2 application menu, but can be dragged to become a floating window or docked to any side of the application window. The toolbar contains short-cuts to some of the commonly-used menu commands. When the Spike2 application is active, and you move the mouse pointer over one of the buttons, the Status bar will display a reminder of the buttons function. If you hover the mouse pointer over a button, a tooltip will appear with a short reminder.

The Toolbar command in the View menu lets you show and hide the system toolbar bar. Script users can show and hide the system toolbar with:

```
View(App(1)).WindowVisible(show%); 'Show%=1 to show, 0 to hide
```

To dock and position the toolbar, use the `Window()` script command.

Status bar



The Status bar occupies the very bottom of the Spike2 application when it is enabled. The left hand end of the bar contains text that describes the currently selected menu command, or a reminder to use F1 to get help. The next field displays the line number and column number of the text cursor when a text view is current, or when a Time, Result or XY view is current it display the last known mouse position as [channel,] x position[, yposition] (items in square brackets are omitted if they do not apply). The remaining fields are:

- CAP** The Caps Lock key is active. Press Caps Lock on the keyboard to disable.
- NUM** The Num Lock key is active. Press Num Lock on the keyboard to disable.
- OVR** Overtyping mode is active in a text-based view so that typing replaces text. Press the Ins key to disable (this only has an effect when the current view is a text-based view).
- REC** Script recording is enabled. You can Turn Recording Off in the Script menu.

The Status bar command in the View menu lets you show and hide the status bar. Script users can hide or show the Status bar with:

```
View(App(2)).WindowVisible(show%); 'Show%=1 to show, 0 to hide
```

Enlarge View Reduce View

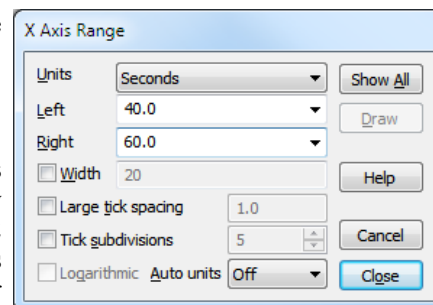
These two commands duplicate the two buttons at the lower left of time and result windows. The enlarge command, short cut `Ctrl+E`, doubles the x axis data region and the reduce command, `Ctrl+R`, halves the region. The left hand window edge is fixed unless enlarging would display data beyond the end of the data, in which case the displayed area is moved backwards. If enlarging would display more data than exists, all the data is displayed. Short cut keys `Ctrl+U` and `Ctrl+I` zoom about the screen centre. You can also change the view size by clicking and dragging the x axis numbers.

X Axis Range

This menu and toolbar command duplicates the action of double clicking on the x axis in a time, result or XY window.

Units

The Units field in a time window selects Seconds, hh:mm:ss (hours, minutes, seconds), Time of day or milliseconds display mode. Time of day works for files created by Spike2 version 4.02 onwards; for older files with no saved creation time it is the same as hh:mm:ss. In result and XY views, the Units field display whatever the X axis units are set to. If the x axis units are "s" or "seconds", you can also choose milliseconds as your units.



The Time of day axis mode draws times as hh:mm:ss on the x axis that are the time of day at which the data was collected plus the time offset into the file. This is for display purposes only; all times used within Spike2 are times in seconds from the start of sampling. All times entered into dialogs are relative to the start of the file.

The milliseconds selection changes how the axis labels are displayed. Internally, Spike2 still uses seconds for all measurements and commands related to the axis. If you select milliseconds, all the time-based fields in the dialog (Left, Right, Width and Large tick spacing) change to milliseconds. When recording actions, if you use expressions (for example `CURSOR(1)+1`) in milliseconds mode that would evaluate to a different result in seconds mode, these will record as the result of the expression in seconds.

Left, Right and Width

The Left and Right fields set the window start and end. You can type in new positions or use the drop down lists next to each field. The drop down list contains the initial field value, cursor positions, the minimum and maximum allowed values and the left and right edges of the window (`xLow()` and `xHigh()`). The Width field sets the window width if the Width box is checked. Click the Draw button to apply changes in these fields to the window. Show All expands the x axis to display all the data and closes the dialog.

If you type a number in this field, it is interpreted in the units set for the Units field. In a time view, you can follow a typed number by s, ms or us to force the number to be interpreted as seconds, milliseconds or microseconds regardless of the settings in the Units field.

Tick spacing

In normal use, you will let Spike2 organise the x axis style. However, when preparing data for publication you may wish to set the spacing between the major tick marks and the number of tick subdivisions. If you prefer a scale bar to an axis, you can select this in the Show/Hide channel dialog.

You can control the Large tick spacing (this also sets the scale bar size) and the number of Tick subdivisions by ticking the boxes. Your settings are ignored if they would produce an illegible axis. Changes made to these fields take effect immediately; there is no need to use the Draw button.

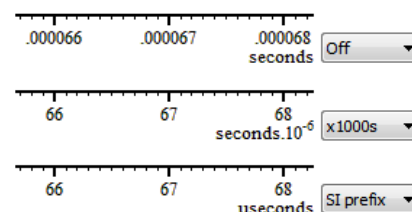
Auto units

The Auto units field is present when the Logarithmic box is not checked and the Units field is not set to milliseconds, hh:mm:ss or Time of day. There are three options:

Off The axis behaves normally, using the standard x axis units.

x1000s If the entire axis in time or result views or the visible axis in an XY view becomes numerically very large or very small and scaling makes sense, the axis units are multiplied by a power of a thousand and the x axis units are displayed with the power of ten after the units.

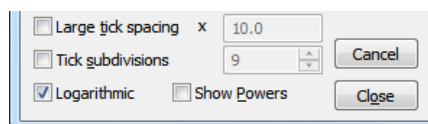
SI prefix The same as x1000s, except that the x axis units are prefixed by a SI scaling prefix (M, k, m, u, ...). If the units already start with an SI prefix, this is removed (and taken into account) before display.



If the units are scaled, cursor labels and on screen measurements (hold down Alt and click and drag) are also scaled. Apart from these cosmetic effects, all other measurements remain in the original axis units.

Logarithmic

The **Logarithmic** check box can be used in Result and XY views to change the x axis to logarithmic mode. In logarithmic mode, the large tick spacing field becomes a multiplying factor between large ticks and is usually set to 10. You can use all drawing modes with logarithmic axes; however, straight or cubic-splined lines between points will not pass through the same values as with linear axes. Furthermore, you can generate apparently blank axes in logarithmic mode. For example, if the data range of the axis was 61 to 69 units, changing this to logarithmic mode will generate a blank axis as the nearest minor ticks are at 60 and 70 and the nearest major ticks are at 10 and 100.



The **Show Powers** check box replaces the Auto-adjust units field in logarithmic mode. Check this box to display the axis labels at major tick marks as powers of 10 (or the value set in the **Large tick spacing** field).

Cancel undoes any changes made with the dialog and closes it. The **Close** button closes the dialog.

Script language equivalents to this dialog

The `XAxisAttrib()` script command is the equivalent of the **Logarithmic** and **Show Powers** check boxes and the **Auto units** selector. The `XAxisStyle()` command sets the **Units** and the tick spacings and number of subdivisions of major ticks. The `XRange()` and `Draw()` commands set the displayed range. You can show and hide axis features with `XAxisMode()`.

Short cut keys and mouse wheel

Short cut keys that control the x axis are: **Home** and **End** move to the start and end of the data, **Left** and **Right** arrow scroll by one pixel, **Shift+Left** and **Shift+Right** scroll by several pixels and **Ctrl+Left** and **Ctrl+Right** move by half the screen width. The mouse wheel will also scroll the x axis one pixel at a time; add **Shift** or **Ctrl** to scroll by larger numbers of pixels.

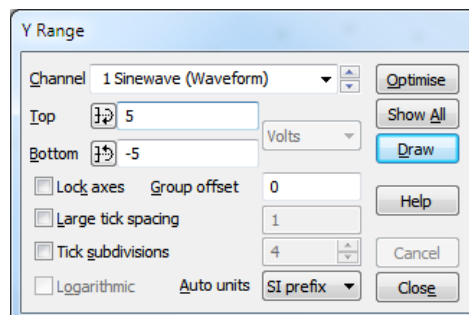
Jump to event

In a time view you can jump to the next or previous event by selecting the event channel, then using **Alt+Right/Left** arrow. Spike2 searches for the nearest event to the centre of the screen to the right or left. If more than one event channel is selected, all channels are scanned for the nearest event. Spike2 beeps if none is found.

You can jump to any **TextMark** from the **TextMark** dialog. Double-click any **TextMark** to open the dialog, click the **>>** button, select the marker from the list and click **Show**.

Y Axis Range

This dialog sets the y axis range and style for time, result or XY view channels. The **Channel** field chooses one, all or selected channels or you can type a list of channels. If more than one channel is selected, the displayed settings are for the first channel that matches the displayed channel units.



Optimise and Show All

Click **Optimise** to draw the visible data at the highest magnification without clipping at the top and bottom. Click **Show All** to set the y axis to display the maximum possible range for waveform channels and from 0 to the estimated event rate for event channels drawn with a frequency axis. Both these buttons close the dialog. You can optimise without opening the dialog with the keyboard short-cut **Ctrl+Q** and by right-clicking on a channel and selecting the **Optimise** option from the context menu. In logarithmic mode, values of zero or less cannot be displayed.

Top and Bottom

The **Top** and **Bottom** fields set the values at the top and bottom of the axis. The buttons to the left of these fields make the axis symmetric about zero. The units of the values are displayed to the right of these fields. If there are multiple channels selected, and the channels have different units, you can select the units or **<any>** to

match any units. Click **Draw** to apply changes to the **Top**, **Bottom** and **Group offset** fields. These changes are only applied to channels that match the displayed units; select **<any>** as the units to apply the values to all channels in the list.

Cancel undoes any changes and closes the dialog. The **Close** button closes the dialog; it does not apply changes to the **Top** and **Bottom** fields.

Lock Axes and Group offset/factor

The **Lock axes** and **Group offset/factor** fields are visible when the current channel shares its y axis with other channels. The fields are enabled when the current channel is the first in the group. If you check the **Lock axes** box, the grouped channels not only share the same space, they also share the y axis of the first channel in the group. The **Group offset** field sets a per-channel vertical display offset to apply to each locked channel so you can space out channels with the same mean level. If the axis is logarithmic the field becomes **Group factor** as a constant shift becomes a multiplicative factor.

Tick spacing and subdivisions

When preparing data for publication you may wish to set the spacing between the major tick marks and the number of tick subdivisions. If you prefer a scale bar to an axis, you can select this in the **Show/Hide channel** dialog. You can control the **Large tick spacing** (this also sets the scale bar size) and the number of **Tick subdivisions** by checking the boxes. Your settings are ignored if they would produce an illegible axis. Changes made to these fields take effect immediately; there is no need to use the **Draw** button.

Auto units

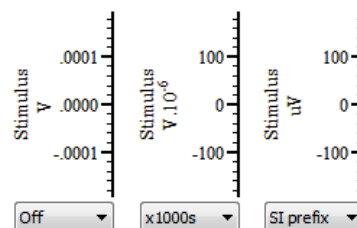
The **Auto units** field is present when the **Logarithmic** box is not checked. There are three options:

Off The axis behaves normally, using the standard x axis units.

x1000s If the axis range becomes numerically very large or very small and scaling makes sense, the axis units are multiplied by a power of a thousand and the x axis units are displayed with the power of ten after the units.

SI prefix The same as **x1000s**, except that the y axis units are prefixed by a SI scaling prefix (M, k, m, u, ...). If the units already start with an SI prefix, this is removed (and taken into account) before display.

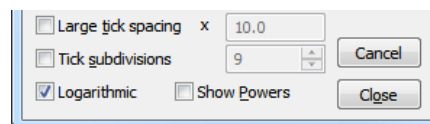
Any change made to an axis by this field is purely cosmetic; it has no effect on the values seen by **Spike2** or by a script. It changes the visible axis values and the values displayed on screen by horizontal cursors and by vertical measurements made by holding down the **Alt** key and clicking and dragging in a channel.



Logarithmic and Show Powers

If the units are scaled, cursor labels and on screen measurements taken by holding down **Alt** and clicking and dragging are also changed. Apart from these cosmetic effects, all other measurements remain in the original axis units.

The **Logarithmic** check box can be used in **Result** and **XY** views to change the y axis to logarithmic mode. In logarithmic mode, the large tick spacing field becomes a multiplying factor between large ticks and is usually set to 10. The **Group offset** field also changes to **Group factor**. You can use all drawing modes with logarithmic axes; however, straight or cubic-splined lines between points will not pass through the same values as with linear axes. If either end of the axis is less than or equal to zero when you set logarithmic mode, the axis limits are adjusted.



The **Show Powers** check box is present in logarithmic mode and replaces the **Auto-adjust units** field. With this box checked, the axis labels at major tick marks are displayed as powers of 10 (or of the value set in the **Large tick spacing** field).

Axis adjustments in logarithmic mode

See the **Edit** menu **Preferences** dialog **Display** tab to set the number of decades to display when an axis switches from linear to logarithmic mode or you **Optimise** in logarithmic mode and the low limit is less than or equal to 0.

Standard Display

This command sets the current time, result or XY view to a standard state. The x and y axes are displayed with all user options for grids, tick spacing and special axis modes turned off. In time and result views, duplicate channels are deleted and all channels are displayed in a standard mode and size and ordered as set in the Edit menu Preferences, all special channel colours are reset and any channel processing is removed. In an XY view, all channels are made visible, the point display mode is set to dot at the standard size, the points are joined and the x and y axis range is set to span the range of the data.

In a time view, Marker derived channels all display the first marker code. WaveMark channels are set to always display markers in hexadecimal format, all others will show printing characters.

In a text-based view it removes any maximum line limit except in the Log view, where it applies the line limit set in the Edit menu Preferences option. It also hides line numbers, displays the gutter and the folding margin (for views that support folding). All the text styles are set to the default values (equivalent to opening the Font dialog and using Reset All) and any zooming is removed.

Show/Hide Channel

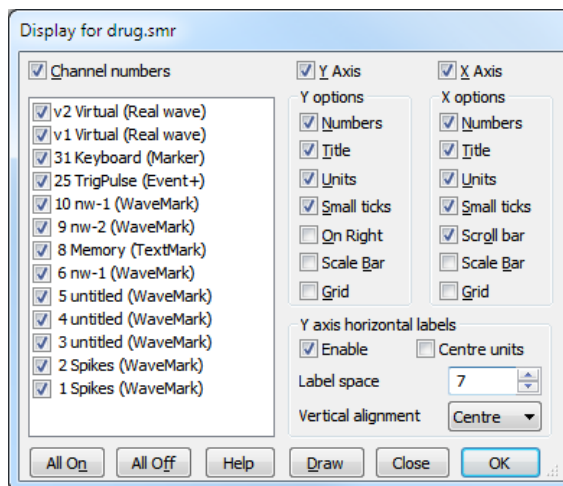
This dialog sets the channel list to display in time, result and XY views. It also controls the display of axes, grids and the horizontal scroll bar in time and result windows.

Channel numbers, Y Axis, X Axis

The three check boxes at the top of the dialog control if channel numbers and the x and y axes are drawn. Channel numbers are never drawn in an XY view as all channels share the same y axis.

Channel list, All On, All Off

The channel list contains all the channels that exist in the view. If there are more channels than will fit in the visible area, you can either scroll the channel list to see them all or resize the dialog by dragging the size box or the top or bottom edge of the dialog. The check boxes in the list are ticked if the channel is visible. You can change the state of the check boxes or use the All On and All Off buttons to set the state of all the channels.



Draw

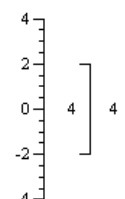
Most changes made in this dialog will require the view to redraw, which can take a noticeable time. Because of this, changes are not applied until you click OK or you can apply changes without closing the dialog with the Draw button.

Y Axis, X Axis

You also have control over the x and y axis appearance. You can hide or display the grid, numbers on the axes, the big and small ticks and the axis title and axis units. You can also choose to show the y axis on the right of the data, rather than on the left and choose if the Scroll bar is visible for the x axis (not in an XY view).

Scale bar

For publication purposes, it is sometimes preferable to display axes as a scale bar. If you check the Scale only box, a scale bar replaces the selected axis and any grid set for that axis will vanish. You can remove the end caps from the scale bar (leaving a line) by clearing the Small ticks check box. The size of the tick bar can set by the Large tick spacing option in the Y Axis Range or X Axis Range dialogs, or you can let Spike2 choose a suitable size for you.



A y axis will automatically switch to draw as a scale bar if there is not enough vertical space to display numbers for the tick marks.

Y axis horizontal labels

Normally, the Title and Unit text for the y axis is presented vertically unless there is very little vertical space available. If you check **Enable**, the y axis text is presented horizontally and the other controls in this region become active. The channel title is left aligned when the axis is on the left and right aligned when the axis is on the right. The script language equivalent of this dialog section is `YAxisMode()`. The other controls are:

Centre units

If checked, the units are centred on the channel title. If clear, the units are aligned to match the title.

Label space

You can set the horizontal character space to reserve for the channel title and units. This is in terms of a representative character width.

Vertical alignment

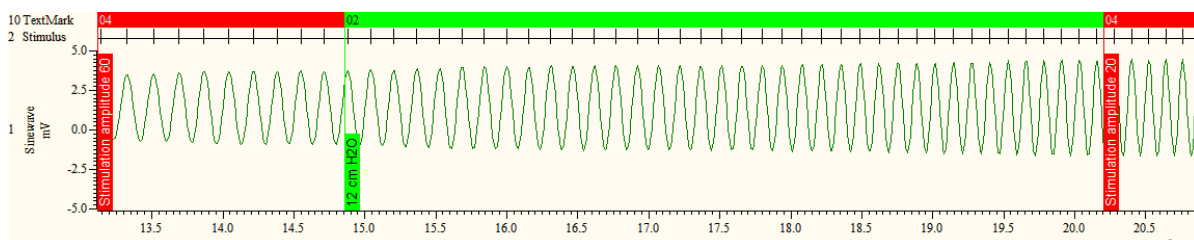
You can choose to position the text at the top, centre or bottom of the available vertical space.

Short cut to display a single channel or channel group

In a time or result view, you can double-click a channel with a y axis to hide all the other channels; the channel expands to fill the entire window. A second double-click restores the display. `Ctrl`+double-click displays the channel plus all duplicates. If the clicked channel holds a group of channels, all channels in the group expand to use the display area and if `Ctrl` is held down, duplicates of all channels in the group are displayed.

Vertical Markers

The View menu Vertical Marker command is available in a Time view and opens the Vertical Markers dialog. Vertical markers are vertical lines drawn under or on top of the channel displays, rather like vertical cursors, except that their positions are set by the times of event or marker data points on a channel. If the channel is a TextMark, you have the option of displaying the text of the marker in a variety of styles. Vertical markers are not drawn if the display is in 3D overdraw mode, nor are they drawn over the Overdraw WM area of a Time view.

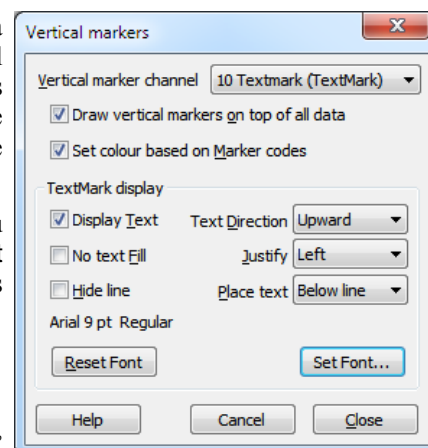


The picture shows an example of the use of vertical markers with a TextMark channel being drawn in both state mode and as a vertical marker. Vertical markers are often used in place of vertical cursors where you need a marker that cannot be moved by the user. If you use a memory channel as the vertical marker channel, you can edit the position by adding and removing items in the memory channel.

You can set the width of the vertical line in the Edit menu Preferences dialog and the default colour in the View menu Set Colours dialog in the Application Colours section. The dialog fields are:

Vertical marker channel

You can select any channel that holds event, Marker, RealMark, WaveMark or TextMark data. You can also select No channel to disable the display of vertical markers.



Draw vertical markers on top of all data

Check this box to draw the vertical markers after the channel data. Leave the box clear to draw it before the channel data. Leaving this box unchecked places the markers on top of any background (including channel

background colours and grids) but under the data. Drawing is faster if this box is checked if you have grids or individual channel background colours defined.

Set colour based on Marker codes

There is a set of 9 colours in the colour palette that are used to display WaveMark data in colours based on the marker code. These colours can also be used for vertical markers if the marker channel is based on a Marker data type. Check the box to enable the use of WaveMark colours.

TextMark display

The remainder of the dialog fields are concerned with TextMark data. If the channel is not of TextMark type all the remaining fields are disabled unless No channel is selected, when all fields can be set.

Display Text

Check this box to display the text stored in a TextMark data channel.

No text Fill

Normally, the area behind the text is drawn in the colour set for the vertical marker and the text is drawn in white or black, whichever provides the most contrast. Check this box to draw the text in the vertical marker colour with no background fill.

Hide line

When text is drawn, you can choose to hide the vertical line by checking this box.

Text Direction

You can choose to run the text upwards (to match the text for channel labels on channels with a y axis) or downwards.

Justify

The text can be Left, Centre or Right justified within the scrollable area of the view.

Place text

The text can be aligned to be Below, On or Above the vertical marker line.

Set Font...

Click this button to open a Font select dialog where you can choose the font to use for the vertical marker text. The current font, size and style displays above the Reset Font button. The font size may be slightly different from the requested size as it displays the actual size of the font as rendered on screen.

Reset Font

Click this button to revert the vertical marker font to the font set for the view. The text above this button will show Use font set for view if the font has been reset.

Cancel

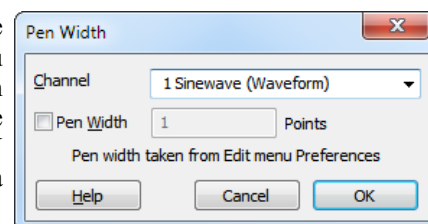
The view changes dynamically to show the effects of any changes. If you close the dialog with the Cancel button, all changes made are removed before the dialog closes.

Close

This closes the dialog, accepting any changes. If you have made a change and recording is enabled, script commands are saved that reflect the new state. You can use Undo to remove changes after the dialog closes.

Pen Width

The Pen Width dialog in a time or result view allows you to override the standard pen width set for data in all channels in the Edit menu Preferences dialog. The dialog can be opened as a context menu when right-clicking on a channel, or from the View menu. Changes made with this dialog can be undone and recorded as a script. For XY views, use the XY Draw Mode dialog to set the pen width of a channel.



Channel

You can select a channel from the drop-down list or type in a channel or a list of channels. If more than one channel is set, the displayed information will be for the first channel in the list.

Pen Width

If you check the box, you can set a pen width in points; this will be applied to all the channels in the list when you click OK. If you clear the box, the standard pen width for data (set in the Edit menu Preferences Display tab) will be applied on OK and you cannot edit the value. A point is 1/72 of an inch, which is about the same size as a pixel. If you set a size of 0, the pen size will be 1 pixel on all devices (which can be very thin on a printer). You can set a fractional point size, but the output pen will always be at least 1 pixel wide.

Info

This command displays information about the current result view window including the number of channels, bins and bin width. In particular, it displays the number of sweeps that have been added into a PSTH, correlation or waveform average, the number of data blocks in a power spectrum, the number of cycles in a phase histogram and the number of intervals that have been processed to build an interval histogram (including intervals that fell outside the histogram).

File Information

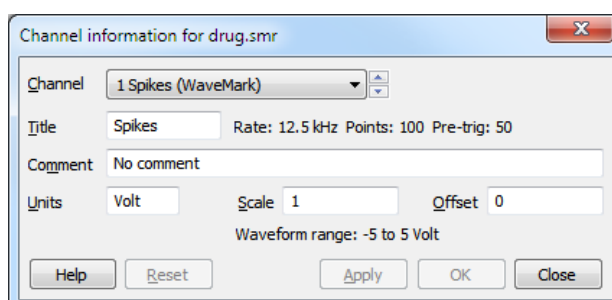
This displays information about the current time view, including five lines of comments and the time and date it was created (if available). You can edit the comments unless the file is open in read only mode. You can cause this window to open automatically when sampling ends from the Automation tab of the Sampling Configuration dialog.

Channel Information (Time view)

Use this dialog to view and edit time view channel information. You can open it with a double-click on a channel title, from the View menu or right click the channel to open the context menu. You can edit the Title and Comment of the channel set by the Channel field. The remaining fields are hidden or displayed depending on the channel type. The Reset, Apply and OK buttons are disabled until you make a change to one of the fields. The Close button closes the dialog and does not apply any changes.

Changes made in this dialog have no effect on your data until you click the Apply button or OK (which is the same as Apply then Close). Changes to all fields except Title apply to all duplicates of the channel and to the channel and its duplicates in any window duplicated from the current window. If you change channel without applying changes, any changes are lost.

Changes to the Title field are applied to duplicate channels if the duplicate has not been given its own



title. If you set the title of a duplicate channel, it has no effect on any other channel. If you clear the title of a duplicate channel, it takes the title of the channel from which it was duplicated.

The **Reset** button restores any changes you have made to the channel settings unless you have used **Apply**. To undo applied changes, close the dialog and use the **Edit** menu **Undo** command (Ctrl+Z).

Scale, Offset and Units

For waveform, RealWave or WaveMark channels, extra fields show the **Scale** and **Offset** values and the channel user **Units**. The scale and offset convert between the 16-bit integers used to store waveform and WaveMark data and user units. They also convert RealWave values to integers, when required.

$$\text{Real value in user units} = 16\text{-bit value} * \text{Scale} / 6553.6 + \text{Offset}$$

$$\text{Integer value} = (\text{Real value in user units} - \text{Offset}) * 6553.6 / \text{Scale}$$

When the **Scale** and **Offset** fields are present, the **Waveform range** field displays the range of values that a 16-bit waveform channel could span. You can read more about scaling in the documentation for the sampling configuration dialog. Both the scale and the offset must be smaller than 10 billion. The scale may not be set to 0 or very close to 0. If the channel scale or offset is edited into an illegal state, a warning message appears in the dialog and you cannot **Apply** the changes. If you want to calibrate a channel, it is often easier to use the **Calibrate** option of the **Analysis** menu.

The **Scale** value is numerically the same as the value in the **Sampling** configuration when data is captured by a 1401 interface that has a ±5 Volt input range. If your unit has a ±10 Volt range, the scale value is double that in the **Sampling** configuration.

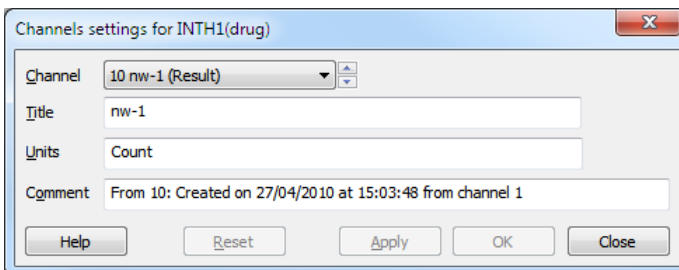
If the **Scale** and **Offset** fields are disabled and "No rescale" appears in the channel information to the right of the **Title** field, you have applied a channel process that has changed the calibration or is non-linear. At the time of writing, **Rectify** and **Fill Gaps** are both non-linear processes. To change the channel scaling, remove the process(es).

Rescale

The **Rescale** button appears for RealWave channels. Click it to set the **Scale** and **Offset** fields so that the full range of data could be represented by 16-bit data. The offset is set to 0 if this does not lose too much precision. Some routines in Spike2 treat RealWave data as 16-bit integer, and the scale and offset you set determine how the conversion from 32-bit real to integer is done.

Channel Information (Result view)

Use this dialog to view and edit result view channel information. You can open it with a double-click on a channel title, from the **View** menu or right click the channel to open the context menu. You can edit the **Title**, **Units** and **Comment** of the channel set by the **Channel** field. The **Reset**, **Apply** and **OK** buttons are disabled until you make a change to one of the fields. The **Close** button closes the dialog and does not apply any changes.



Changes made in this dialog have no effect on your data until you click the **Apply** button or **OK** (which is the same as **Apply** then **Close**). Changes to all fields except **Title** apply to all duplicates of the channel and to the channel and its duplicates in any window duplicated from the current window. If you change channel without applying changes, any changes are lost.

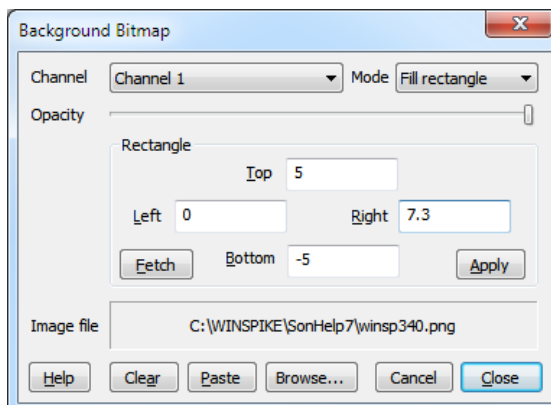
Changes to the **Title** field are applied to duplicate channels if the duplicate has not been given its own title. If you set the title of a duplicate channel, it has no effect on any other channel. If you clear the title of a duplicate channel, it takes the title of the channel from which it was duplicated.

The **Reset** button restores any changes you have made to the channel settings unless you have used **Apply**. To undo applied changes, close the dialog and use the **Edit** menu **Undo** command (Ctrl+Z).

Channel Image

You can set a bitmap as the background to any channel in a Time, Result or XY view. This can be very useful in an XY view when the XY view is being used to map activity. For example, when the view is showing where spiking activity occurred in a maze, or when tracking a target. The menu command opens a dialog:

The dialog is used to select a channel and then link the channel to a bitmap file on disk. Any channel in the view can be associated with an image. The name of the file appears in the Image file section of the dialog. The image is rendered on top of the channel background colour, but below all other items that are drawn. When channels are overdrawn, the image appears when the background colour of the channel would be drawn. The dialog fields are:



Channel

You can select any channel in the current view. However, if you are in an XY view, all channels share the background, so the choice of channel makes no difference. When you change channel, any **Rectangle** region changes that have not been applied are lost and the current settings for the new channel appear in the dialog.

Mode

There are three mode settings: **No display**, **Fill Background** and **Fill Rectangle**.

- No display** No image is displayed. This allows you to have an image loaded, ready to display. This will save the time needed to load the image from a disk file.
- Fill Background** The image is scaled so that it fills the available channel rectangle. You would use this when you want to display the entire image as a background for the channel. We have not optimised the display of background images in this mode for Time and Result views when the view scrolls and suggest that you avoid this combination as the visual effect can be unpleasant - we may decide to remove this mode for Time and Result views). We disable background images in **Fill Background** mode when sampling or rerunning.
- Fill Rectangle** The image is scaled so that it fills a rectangle defined in x and y axis units. You would use this when the image must be aligned with the axes. For example, if the x and y axes represented co-ordinates in a maze, the image could be a picture of the maze. There is no problem with scrolled displays in this mode.

Changes made to the mode field are applied immediately, so you can see the effect.

Opacity

This field is a slider control that you can drag to the left to make the image transparent and to the right to make it opaque. Changes made by dragging are applied immediately.

Rectangle

The fields within this area are disabled unless the mode field is set to **Fill Rectangle**. The **Left**, **Right**, **Top** and **Bottom** fields refer to the x axis and y axis co-ordinates of the image. In the current implementation, you cannot invert or reflect the image; if you do it will not appear.

Fetch

This button sets the **Left**, **Right**, **Top** and **Bottom** fields to the values that would make the image fill the channel background.

Apply

This button is enabled when in **Fill Rectangle** mode and the current rectangle settings are legal number and do not match those of the channel. Click the button to apply the new values.

Browse...

Click this button to open a file dialog in which you can browse for a suitable image file. You can choose between windows bitmaps, JPEG images, PNG files and TIFF files.

Paste

This button is enabled if the clipboard holds a bitmap image. Click to paste the image to the current channel. The Image file name will display as <CB>. Setting the clipboard as the source can have unexpected results as each time a saved window opens it will copy whatever image happens to be on the clipboard. Clipboard images are more useful to scripts, for example to copy a multimedia view video frame using `EditCopy()`, then setting the image with `ChanImage()`, avoiding the need to write the image to a disk file.

Clear

Click this button to clear any image set for the channel. This releases any resources used to hold the bitmap.

Cancel

Click this button to close the dialog and undo any changes made since the last channel change.

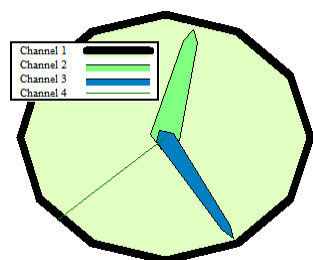
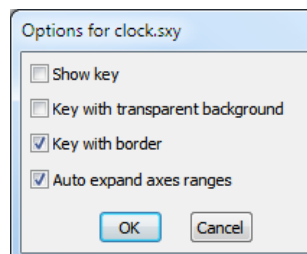
Close

Close the dialog, leaving the associated channel in the current state.

This command is experimental and is implemented in a simplistic manner. If you reuse the same image for multiple channels we do not notice, and store the image multiple times, which could be a problem if you use huge images on many channels.

Options

This command is for XY windows only and opens the XY options dialog. This dialog controls the XY window “key”. The key is a small region to identify the data channels that you can drag around within the XY window. For each visible channel it displays the channel name and draws the line and point style for the channel. This dialog also has a check box that controls the automatic expansion of the axes when new data is added. The equivalent script language command is `XYKey()`.

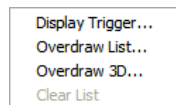


Scripts\clock.s2s

This example XY view, (generated by `clock.s2s` in the `Scripts` folder) shows the key. In this example, several channels use filled mode. The channel fill colour displays below the line in the key. You can set the key background transparent or opaque and choose to draw a border around the key. If you move the mouse pointer over the key, the pointer changes to show that you can drag the key around the picture. Double-click the key to open the Options dialog.

Trigger/Overdraw

This View menu command is for time views only and opens a pop-up menu that leads to commands for controlling triggered displays, display overdrawing and 3D (three dimensional) displays of sweeps of data identified by trigger events:



- Display trigger** This is the overall setup dialog for triggered and overdrawn displays. Overdrawn displays use a stored list of trigger times.
- Overdraw List** A quick way to overdraw using an event channel as a source of multiple trigger times.
- Overdraw 3D** Controls how frames are shifted and scaled to give a 3D effect.

Clear List

Empty the stored list of trigger times.

Display Trigger

The display trigger is used with Time views to provide an oscilloscope style trigger, paged display on-line, display overdraw, a 3D display and a means of easily moving back and forward to the next or previous trigger both on-line and off-line. The script language equivalent of this command is `ViewTrigger()`.

Trigger channel

The Trigger channel field sets an event, Marker, WaveMark, TextMark or RealMark channel to be used as the trigger. You can select Paged display for a permanent trigger (for on-line paged displays). This field can also display No channel is selected if the overdraw trigger list contains times from a mix of data channels.

Pre-trigger display time

The Pre-trigger display time field sets the time before the trigger to show each time a trigger occurs. This dialog does not set the width of the time view; that is set by the normal time view mechanisms. If you set the pre-trigger display time larger than the displayed time view width or negative, the trigger point will not be visible. Negative pre-trigger times move the trigger point off the screen to the left of the display.

Minimum display hold time

The Minimum display hold field is used on-line and sets the minimum time that data is displayed after the current time passes the right-hand edge of the screen. This allows you to see individual data frames with a high frequency trigger. A value of 0 means wait until the current screen is displayed before looking for new triggers.

Cursor zero action

The Cursor zero action field has three setting that control what happens to cursor 0 and any active cursors that depend on it when the view triggers:

- | | |
|------------------|--|
| No action | Cursor 0 state is unchanged. |
| Move to trigger | Cursor 0 moves to mark the trigger point, active cursors do not move. |
| Move and iterate | Cursor 0 moves to mark the trigger point, active cursors 1-9 move. The Hold off iteration field is made visible. |


Hold off iteration

This field appear when Cursor zero action is set to Move and iterate. It is used online and sets a time, in seconds, to delay the active cursor iteration after the trigger. This allows you to search for data features after the trigger point when sampling or rerunning (when data after the trigger has not yet been sampled).

Zero x axis at trigger

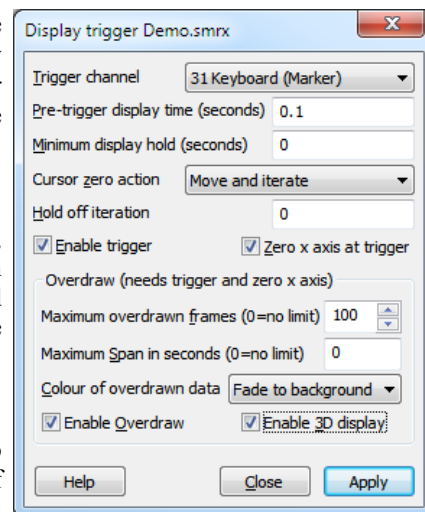
The Zero x axis at trigger check box changes the x axis so that 0 lies at the trigger time. This is purely a visual convenience; all measurements are still in the original x axis units. You must check this box if you want to use display overdraw or 3D display modes.

Enable trigger

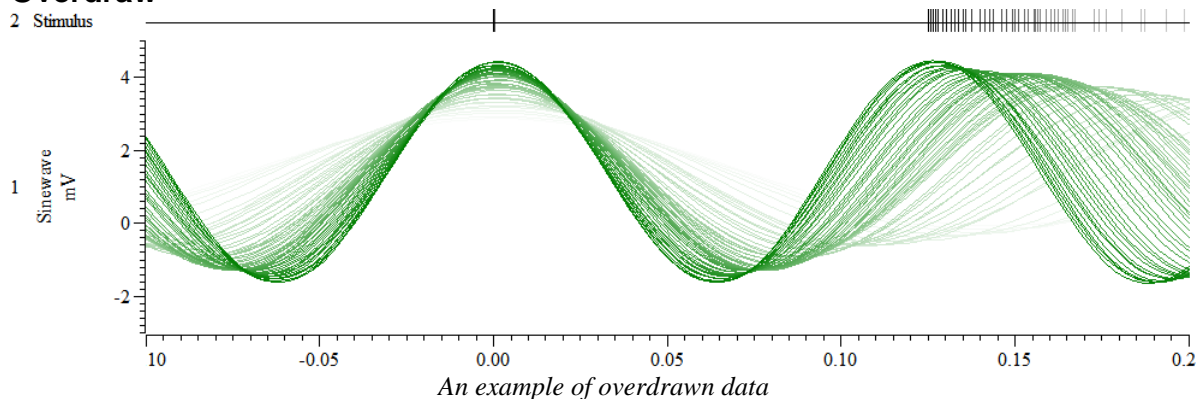
Check the Enable trigger check box and click OK to allow display triggering. Two extra buttons  appear in the area to the left of the x axis scroll bar. These buttons can be used on-line and off-line to move backwards and forwards by a trigger. With Paged display set, they move by the time between the trigger point and the right hand screen edge.

Keyboard control

In addition to clicking the buttons, you can also step to the next and previous trigger point with the Alt+Shift+Right and Alt+Shift+Left key combinations.



Overdraw



The overdraw section of the dialog is enabled when both **Enable trigger** and the **Zero x axis at trigger** are checked. In overdraw mode, data sections identified by triggers are drawn over each other in time order, oldest first up to the current time. We call each overdrawn section a *frame*. You can change the displayed time range as normal, but trigger times are ignored if time before 0 or time after the end of the file would be displayed. Each new trigger time is added to a list of previous times. If you step backwards, all trigger times after the time you step to are forgotten. You can also add trigger times with the **Overdraw List** dialog. Accepting the dialog settings with **OK** clears the list if the new settings are incompatible with the old settings (for example if the channel is changed).

Overdraw is just a display mode. All measurements, cursor positions and the like behave as if the overdrawn frames are not present.

Maximum overdraw frames

You can limit the number of overdrawn frames of data. Values up to 4000 are allowed, or 0 meaning no limit on the number of frames. It takes time to draw each frame, and you will experience significant screen update delays if you display huge numbers of overdrawn frames of data over long time periods. You can break out of such long displays with the **Ctrl+Break** key combination. If you set a limit, this can be used in 3D mode to set a z axis of constant length. We suggest that you set a limit to the number of frames, particularly for on-line use, to limit the drawing time.

Maximum span in seconds

You can also limit the time range of the overdrawn frames with this field. Each time you add a trigger time, all trigger times after the new time and any trigger time that is more than the Maximum span before are discarded. You can set the value 0 for no time limit. If you set a limit, this can be used in 3D mode to set a z axis of constant length.

Colour of overdrawn data

The last frame drawn (which corresponds with the current trigger time, that is the time for which the x axis is showing 0) is always in the standard colour. You can choose how the remaining frames are drawn from:

No change	Draw in the normal colours.
Half intensity	A equal mix of the normal colour and the background colour.
Fade to background	A gradual fade from the normal to the background colour. In 3D mode, the colour depends on the z axis value, otherwise it depends on the frame number.
Fade to secondary	A gradual fade from the normal to the secondary colour. If no secondary colour is set, this is the same as Fade to background .

Enable Overdraw

Check this box to enable overdraw mode. You must also have checked **Enable trigger** and **Zero x axis at trigger**. Frames are added to the overdraw list by stepping to the next trigger event or with the **Overdraw List** dialog or the `ViewOverdraw()` script command.

Enable 3D display

Instead of overdrawing the frames of data exactly on top of each other, you can choose to draw them offset both vertically and horizontally to create a three-dimensional effect. Check this box to enable this drawing method. The 3D drawing only occurs if you have also checked **Enable trigger**, **Zero x axis at trigger** and **Enable Overdraw**. The screen arrangement of channels and frames in 3D drawing mode is controlled by the

Overdraw 3D dialog.

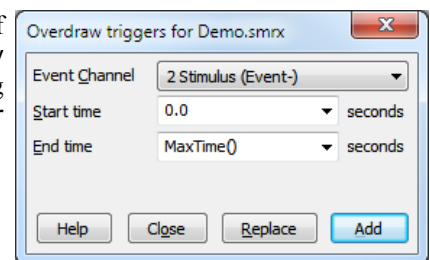
On-line and Rerun use of triggered displays

When enabled, incoming trigger channel data is searched until a trigger is found when the display will hold with the pre-trigger time shown before the trigger until another trigger occurs. The hold time will be at least as long as the Minimum display hold field from the point where the display reaches the right-hand edge of the screen. In **Overdraw** mode, the display hold time limits the number of frames that will be overdraw as if you trigger at time t , the earliest time the next trigger will be used is: $t + w - p + h$ where w is the screen width, p is the pre-trigger time and h is the display hold time.

With **Paged display** selected, sampling begins as normal, then each time the right-hand screen edge is reached and the hold time has passed, a new sweep starts. The pre-trigger time sets the overlap between the sweeps. In overdraw mode, each new sweep adds a new trigger time.

Overdraw List

In a time view you use this **View** menu command to add a list of trigger times to a time view based on the settings in the **Display Trigger** dialog. If the time view is not in overdraw mode, the dialog displays a warning message and a button to open the **Display Trigger** dialog where you can set overdraw mode. Dialog controls are:



Event Channel

An Event, Marker or Marker-derived channel in the time view to use as a source of trigger times to add to the list held by the time view.

Start time, End time

The time range to search for events to add to the list. All events found in the time range are used as trigger times.

Replace

Click this button to clear any existing times from the list before adding new times. If the **Event Channel** is not the same as the channel set in the **Display Trigger** dialog, the **Display Trigger** dialog channel is changed to match.

Add

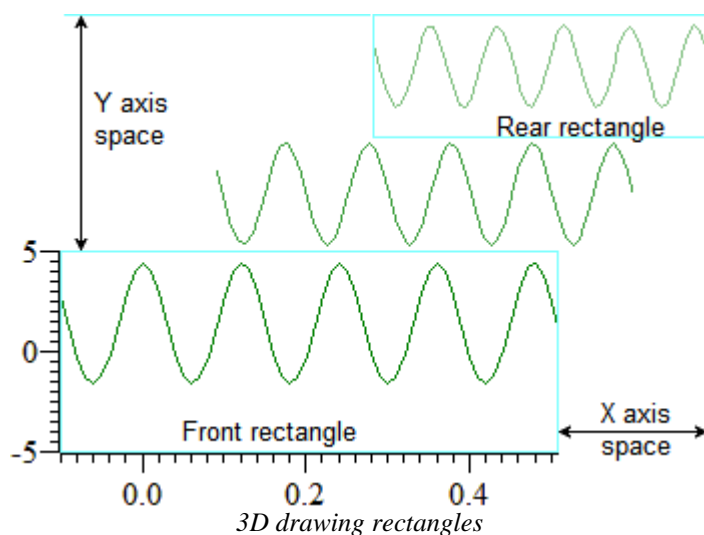
Click this to add times identified by the channel and the time range to the list. Unlike manually adding times by stepping to the next or previous event when the last added time causes all later times to be deleted, added times are merged into the list in ascending time order. The current time display is set to be the last time in the list.

Display Trigger dialog interaction

If you add events to the trigger list such that the event times come from a mix of channels (or if you add times not associated with a channel using a script), the **Display Trigger** dialog channel will change to **No channel is set**. If you replace events such that the trigger list holds times from a single channel, the **Display Trigger** dialog will show that as the current channel.

Overdraw 3D

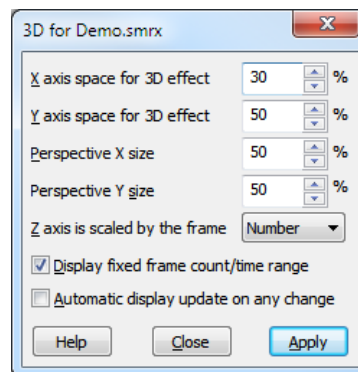
In a time view you use this View menu command to open the 3D dialog to control the 3D drawing effect. The script language equivalent is `ViewOverdraw3D()`. This dialog does not enable 3D drawing; use the Display Trigger dialog to do that. To create the 3D effect, each frame is drawn inside a rectangle and the position and size of the rectangle depends on a notional z axis. The z axis is based on the frame number or the trigger time of the frame. The z axis can be set to a constant frame count or time range, or it can vary depending on the current list of frames to display.



In this example, there are three

frames of data. The *Front* rectangle, used for the most recent trigger, is always positioned at the bottom left of the channel area. The *Rear* rectangle, used for the oldest trigger time, is always positioned at the top right of the channel area. The middle frame also has a rectangle that is calculated by a linear interpolation between the front and rear rectangles based on the z axis position of the frame.

The dialog fields control the positioning of the front and rear rectangles:



X axis space for 3D effect

This field sets the percentage of the width of the channel area to use for the 3D effect. The larger the value, the smaller the width of the front rectangle. Set this and the *Perspective X size* fields to 0 to make all frames draw vertically aligned.

Y axis space for 3D effect

This field sets the percentage of the entire view vertical space to share between all the channels to generate the 3D effect. All channels are given exactly the same space so that the 3D effect is the same for all channels. The larger the value, the smaller the height of the front rectangle.

Perspective X size

This sets the width of the rear rectangle as a percentage of the width of the front rectangle in the range 0 to 100. Setting a value less than 100 gives a perspective effect.

Perspective Y size

This sets the height of the rear rectangle as a percentage of the height of the front rectangle in the range 0 to 100. Setting a value less than 100 gives a perspective effect.

Z axis is scaled by the frame...

The position of each frame of data (between the *Front* rectangle and the *Rear* rectangle) can be set either by the frame number (giving equally spaced frames), or by the frame trigger times. Choose from **Number** or **Time**.

Display fixed frame count/time range

If you check this field and you have a set a maximum number of overdrawn frames (for z axis scaled by **Number**) or a maximum time span (for z axis scaled by **Time**) in the **Display Trigger** dialog, the z axis will be of a fixed size. If you do not check this box, or there is no maximum number of frames or the time range set, the length of the z axis is taken from the number of frames in the list or the time span of frames in the list.

Automatic display update on any change

If you check this box, and change made to the dialog will cause the display to update. If your display does not take long to update you probably want to check the box, but if you have a lot of data to draw you may prefer to update with the Apply button.

Notes

The 3D display mode behaves in exactly the same way as the Overdraw mode except that the overdrawn frames are offset and scaled. There is one other difference; if there are no frame trigger times in the list, then nothing is displayed. There is no restriction on the drawing modes you can use (other than your own common sense). Overdraw WaveMark and Sonogram modes are unlikely to be useful.

The 3D display mode makes no difference to any measurements you may make. There is still a current time range that is displayed (this corresponds to the frame of data that is at the front of the display).

Clear List

This command from the View menu Overdraw/Trigger pop-up menu removes all stored times from the list used for overdrawing.

Time view overdraw details

Each time view maintains a list of overdraw times. The list of times is always held in sorted time order with the smallest time first and the largest time last. There are no duplicated items. The size of the list can be limited to a maximum number of items and to a maximum time span between the first and the last item. If you do not limit the size or time range and the list becomes very long, it can take a long time to draw the data. You can break out of drawing with the `Ctrl+Break` keyboard command.

Adding items to the list

Items can only be added to the list when the time view is in overdraw mode. There are two modes used when adding times:

- Normal** This is the mode used when you step interactively between trigger times using the keyboard commands `Alt+Shift+Left/Right` or use the trigger step buttons at the bottom of the time view. The new time is added to the list (if it is not already present), and all later times in the list are deleted. The added time becomes the current time, the x axis is set to display 0.0 at this time and the display is updated to show the pre-trigger time before the new event. If there is a time range set or the list is full, the first (the oldest) time in the list is deleted to make room for the new event.
- Merge** This is the mode used when you use the **Overdraw List** menu command to add a list of times from an event channel. New times are merged into the list in time order. If a time range is set and the new time causes other times to be out of the time range or if the list is full, the event in the list furthest away in time from the new time is deleted to make room. The last item in the list sets the current time for drawing as described for **Normal** mode.

Users of the script language `ViewOverlay()` command can choose the mode to add times.

Source channel changes

Normally, the trigger times are taken from a single channel. If this is the case, the commands to step to the next and previous event (`Alt+Shift+Left/Right`) work as normal; each step adds the time of the event to the list as described for **Normal** mode. However, if you add events to the overdraw list from a mix of channels, or use the script language to add a list of times (so there is no channel), these commands step backwards and forwards through the times in the list without adding or deleting any times and any previously set channel is ignored. The `ViewTrigger(-1)` command returns -1 if there is a list, but no channel.

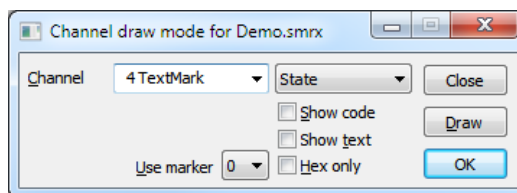
Channel Draw Mode

This menu item is available when the current window holds a Spike2 data document or result view and sets the data channels display mode. You can set the mode for a single channel, all channels, or selected channels. The dialog changes, depending on the channel type and display mode. The Draw button is common to all modes and updates the display without closing the dialog box.

There are different controls for Time views, Result views and XY views.

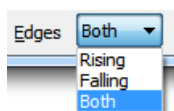
Time view drawing modes

The top line of the dialog is always the same and holds controls to select the channel and the drawing mode to apply. You can select multiple channels by typing in a channel list, selecting All channels or by selecting channels in the time view and choosing Selected. If you select multiple channels, the displayed settings are from the first channel in the list.

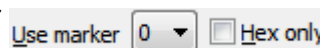


The Close button shuts the dialog without making any change, Draw applies the current settings and leaves the dialog open. OK accepts the current settings and closes the dialog. The remaining controls vary with both the channel type and the drawing mode.

In some modes the Edges field appears if you choose a channel of event level data. You can select Rising, Falling or Both edges of the data. If you select both edges, then the display modes that show frequency count both the rising and falling edges of the event signal in their rate calculations. You would normally count only one edge, so select the edge you prefer.



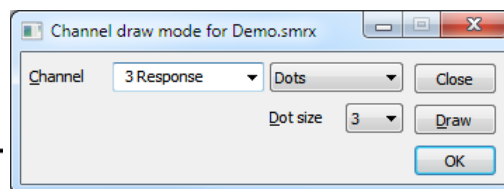
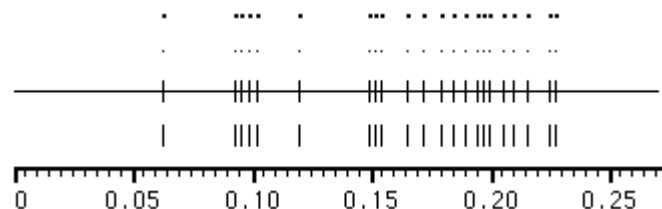
For Marker, RealMark, TextMark and WaveMark channels, the Use marker and Hex only fields may appear. The Use marker field appears if the code is displayed or sets the colour of the items. Normally code 1 is used, but you can choose any of the four codes. If you select a code other than 1, the channel number field will display in red as a warning of a non-standard display mode.



The Hex only field appears if the marker code is displayed. Check the box to display all codes as two hexadecimal digits, unchecked to display codes 0x20 to 0x7e as single characters. The View menu Standard Display command sets Marker code 1 and the appropriate Hex only setting for the channel type.

Dot size 3 If the display will show the data as dots, the Dot size field appears and you can set any dot size from 0 (the smallest mark possible) to 1 to 9 times the thickness of the lines set for drawing data in the Display tab of the Edit menu Preferences. If you set a size to match or exceed the Round dot minimum size field in Edit menu Preferences, the dots will be drawn as circles. Round dots take longer to draw than square dots.

Dots and Lines mode

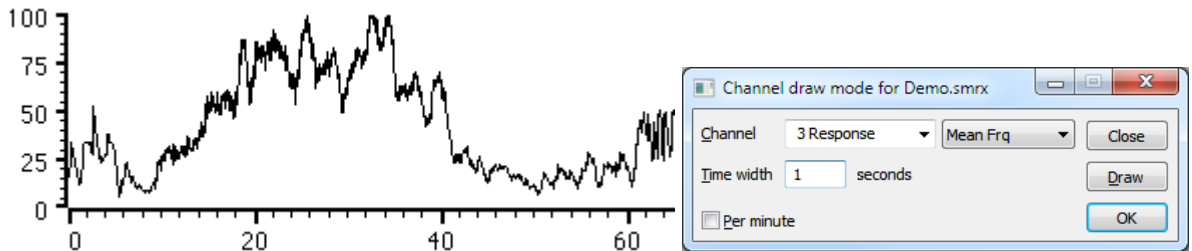


The simplest event channel draw method is dots. You can choose large or small dots (small dots can be very difficult to see). You can also select Lines in place of Dots. The picture shows the result of both types of display on an event channel.

Marker channels displayed as dots also show the currently selected marker code. If you select lines for a marker channel, the display of marker codes is suppressed. In Lines mode you have the option to suppress the central horizontal line.

You can also select Dots mode for a waveform channel.

Mean frequency



The mean frequency is calculated at each event by counting the number of events in the previous period set by Bin size. The result is measured in units of events per second unless the Per minute box is checked. The mean frequency at the current event time is given by:

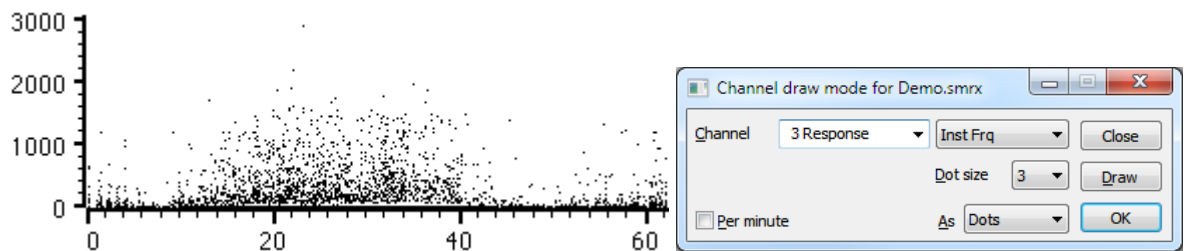
$$\begin{aligned} (n-1)/(te-tl) & \quad \text{if } (te-tl) > tb/2 \\ n/tb & \quad \text{if } (te-tl) \leq tb/2 \end{aligned}$$

where:

tb is the bin size set, te is the time of the current event, tl time of the first event in the time range and n is the events in the time range

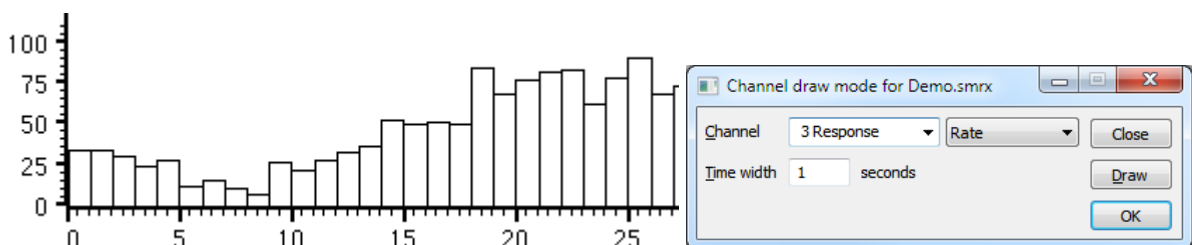
A constant input event rate produces a constant output until there are less than two events per time period. You should set a time period that would normally hold several events.

Instantaneous frequency



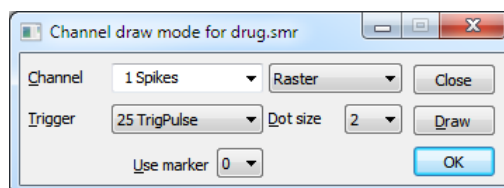
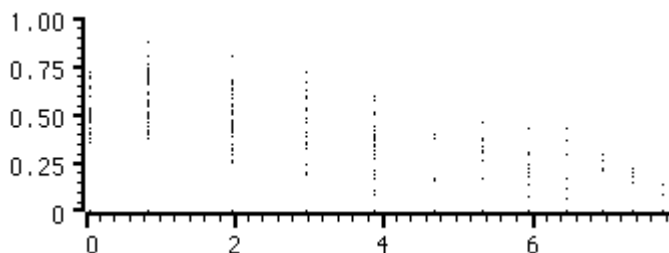
Instantaneous frequency is the inverse of the time interval between an event and the previous one on the same channel. Check the Per minute box for a rate per minute rather than per second. You can display the result as Dots, Line (linking the dots) or Skyline (horizontal lines between dots). In Dots mode you can choose the dot size. You can display the frequency as events per minute rather than per second.

Rate histograms



Rate mode counts how many events fall in each time period set by the Time width field, and displays the result as a histogram. The result is not divided by the bin width. This form of display is especially useful when the event rate before an operation is to be compared with the event rate afterwards.

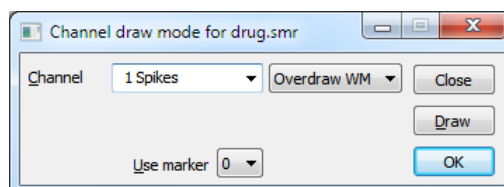
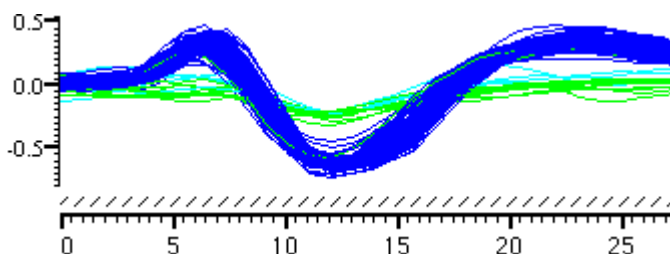
Raster display



Raster mode shows the event positions relative to trigger times. Each trigger event defines time 0 in the y direction for the sweep. The Y Range dialog sets the time range to display in the y direction; negative times show pre-trigger events. For each trigger, events are drawn no further back in time than the previous trigger and no further forward in time than the next.

If the channel is marker-based, events are drawn in the colours set for the selected WaveMark code.

Overdraw WM



Overdraw WaveMark mode draws WaveMark data as superimposed waveforms. Channels drawn in this mode are moved to the top of the window and separated from the x axis (which does not apply to them) by a hatched bar. If this mode is used during data capture and the screen is scrolling to show the latest data, new WaveMark events are added, but old events are not erased (to stop the display flickering). Click on the x axis scrollbar thumb to force an update.

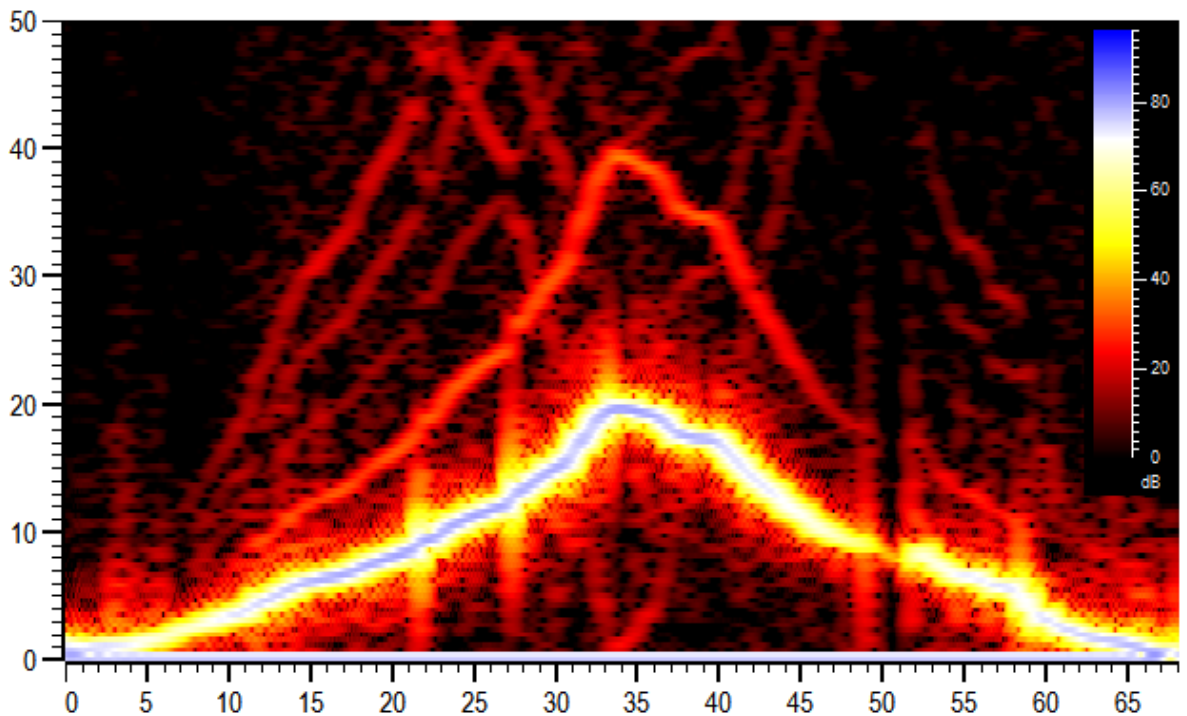
Find with cursor 0

To locate an event, right-click on its waveform at a point where it is clear of all other waveforms and select Find with cursor 0 in the pop-up menu. This locates the event nearest to the clicked position and moves cursor 0 to it. If the Edit WaveMark dialog is open, this will display the event. Any active cursors will iterate.

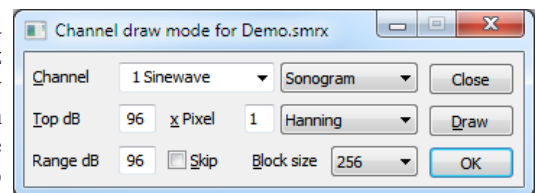
Set WaveMark codes

Hold down **Alt+Ctrl** and click and drag a line over the events you want to identify. On mouse up, a dialog opens in which you can set codes for the intersected events. If the channel has multiple traces the line must only extend over a single trace.

Sonogram display



Sonogram mode shows how the frequency content of a waveform channel changes with time. The y axis units are Hz (frequency) and useful results are available for the frequency range 0 to one half of the sampling rate for the waveform channel. You can control the colour scale used for the sonogram and display a key to indicate how the colours map to intensity.



By default, intensity of the frequency content is indicated by a grey scale, the darker the image, the more intense the signal. However, you can choose a colour scale, or create your own scale in the Edit menu Preferences Display tab. You can set:

Top dB This sets the signal intensity that maps to the top of the sonogram colour scale. dB means decibels, which is a logarithmic measure of ratio, usually of amplitudes or power. 20 dB is a factor of 10 in amplitude. Spike2 stores waveform data as 16-bit integers, and we measure the amplitude with respect to 1 bit, so 96 dB is the maximum possible level. For RealWave channels, the range is much greater, but the 0 dB is still taken as the value that corresponds to 1-bit if the data were converted to 16-bit integer using the current scale factor. That is the 0 dB level is $scale/6553.6$ where $scale$ is the channel scale factor.

If you display the sonogram key you can choose the signal amplitude that is drawn as 0 dB in the key. This is purely a convenience for interpreting the data and has no effect on how the colour map is mapped to the intensity of the signals.

Range dB This sets the range of data to display as a colour map. Signals with an intensity of Top dB - Range dB (or less) are displayed as the colour at the bottom of the sonogram colour scale. If you are unsure what dB values to set for a new signal, setting values of 96 dB for both Top dB and Range dB will display something in almost all cases!

You can be caught out with RealWave data channels. If you are not getting sensible results, double click on the channel y axis to open the Channel information dialog, then click the Rescale button. This will set an appropriate channel scale factor for almost any channel.

x Pixel You can speed up drawing, at the expense of resolution, by setting this field to values greater than 1. It sets the number of screen pixels in the x direction to calculate at a time. A value of 1 gives the best visual resolution (and the slowest calculation and drawing time).

Window The sonogram is calculated using a Fast Fourier Transform. As explained in the analysis section

for the Power spectrum, it is important to apply a “data window” to the signal before taking the power spectrum, otherwise the results are difficult to interpret. We provide several different types of window: None, Hamming, Hanning, Kaiser 30dB, Kaiser 40dB, Kaiser 50dB, Kaiser 60dB, Kaiser 70dB, Kaiser 80dB and Kaiser 90dB.

All windows are a compromise between increasing the apparent width of a spectral peak and the ability to see small signal peaks in the presence of large ones. If you apply no window, you will get the sharpest resolution of a single peak. However, you will not be able to see any small peaks around it due to the “side lobes” of the window. If you are not familiar with the use of windows, the Hanning window is a reasonable compromise. The Kaiser *xx*dB family of windows has the property that the largest side lobe is *xx* dB below the peak. Of course, the larger the *xx* dB, the wider the peaks become.

Special calculations are done where there are discontinuities (gaps) in the data and at these points we ignore the selected window to reduce the computational effort. This effect persists up to half the FFT block size around each discontinuity.

Block size This determines the number of data points used in the FFT, and thereby it determines the frequency resolution. Like the choice of data windows, this is also a compromise. The larger the block size, the better the frequency resolution, but the worse the time resolution. If you are looking for a short, localised burst of changing frequency, you will need to use a block size that is smaller than the duration of the episode you are looking for.

Skip If you are analysing a lot of data, there can be many thousand data points for each screen pixel. If you check this box, the sonogram will only analyse the first **Block size** points for each pixel (normally the sonogram analyses all the points). This can save time if you have a really large file. Of course, the result will only represent a “sampling” of the correct response.

You can export Sonograms as enhanced metafiles and as bitmaps. Printing is supported to Postscript compatible printers. With other printer types, the intensity scale output may be quite coarse. You may obtain better output by saving the sonogram as a bitmap and printing from specialist bitmap editing programs.

Sonogram key

To display the sonogram key, right-click on a channel displayed in sonogram mode and select **Show Key** from the context menu. The key is a rectangular window that can be dragged around the time view. It holds a colour bar with an axis, mapping the colours in the sonogram display to dB. Interactive key actions are both recordable and can be undone with `Ctrl+Z`. The key can be controlled from a script with `ChanKey()`.

Key position

The key is normally positioned on top of the channel that it relates to, but you can drag it wherever you like in the data area of the time view. When dragging the key, you will notice that if you drag it out of the channel rectangle, it is slightly “sticky” at the upper and lower edges of the channel rectangle. This is to let you align it more easily with the top or bottom of the channel.

The key position is remembered either relative to the channel rectangle, or relative to the view data area. Which it is relative to determines how the key positions itself if the view is resized or the number of displayed channels changes. Normally, if you drag the key so it lies within the channel rectangle, the position is remembered relative to the channel, otherwise it is relative to the view. However, if you hold down the `Ctrl` key and drag, the position is always relative to the view and there is no “stickiness” when you cross a channel edge.

Context menu

If you right-click on the key a context menu appears with several key-specific options.

Hide Key

This removes the key from the display. You can restore the key with all its settings unchanged by using the channel **Show Key** context menu command.

Rotate Key

The sonogram key can be displayed with a horizontal or a vertically axis. This context menu command switches between the two possible states.

Default background

The key can be displayed with the background set to the sonogram colour for the lowest intensity signal, or you can select this option to set the background to the background colour set for the data view.

Set dB offset

This opens a dialog that lets you set the signal value that is used as the 0 dB reference level for the axis.

Default position

Moves the key to align with the top-right of the channel rectangle.

Sonogram colours

Opens the Sonogram Colours dialog. Currently all sonograms share the same colour map.

Sonogram key dB offset

This dialog lets you set the 0 dB reference point for the axis in the sonogram key. It has no effect on how the sonogram data is displayed (this is determined by the Draw Mode dialog).

When you display the sonogram key, the axis is graduated in dB, and you will probably find that the 0 dB level is somewhat arbitrary (being relative to 1-bit of the equivalent 16-bit waveform). This is especially the case if your source data is a RealWave where the concept of 1 bit does not apply (but see below). This dialog allows you to choose a reference level in terms of the channel data. For example, if your data is calibrated in Volts, you may want the 0 dB point to be referenced to 1 Volt rather than 1 bit. To do this, just type in 1 in the edit box and hit OK.

To set the key back to reference 1 bit you can type 0 in the edit box and click OK or click the Reset button, which resets the scale and closes the dialog.

Sonogram details

This section contains more technical information about the sonogram and how it is implemented.

Db scale

Sonograms show a representation of the power in a signal using a map of colour to intensity. The dB (deciBell) scale is used for this purpose. This scale describes a quantity (in our case power) with respect to a reference value. It is calculated as:

$$\text{dB} = 20 \cdot \log(A/A_r)$$

where $\log(x)$ means log to the base 10 of x , A is the amplitude of a quantity and A_r is the reference amplitude. An alternative representation is:

$$\text{dB} = 10 \cdot \text{Log}(P/P_r)$$

where P is a power and P_r is a reference power. As power is proportional to amplitude squared, it is easy to see that these definitions are equivalent.

The 1-bit reference

Spike2 originally supported 16-bit integer waveform data only (in the range -32768 to 32767). Scale and offset values convert the integer data into user values. The sonogram was set, by default, to display data with respect to an amplitude of 1 data bit. Unless you use the Draw Mode dialog to adjust things, the top of the scale is mapped to 96 dB (relative to 1 bit) and the range of the display is set to 96 dB. With 16-bit integer input the maximum input amplitude for an unclipped signal relative to 1 bit is 90.3 dB ($20 \cdot \log(32768)$). If there is random noise present, you can visualise amplitudes smaller than 1 bit, but for most purposes, the default settings span the useful 16-bit waveform range. However, then we added RealWave data channels.

RealWave data

RealWave channels store data as 32-bit floating point. These channels have a scale and offset value that is used (when required) to convert the data to integer values. Thus, there is still the concept of 1-bit of integer data,

and this is used as the reference. However, if the scale value is poorly chosen, (either far too big or far too small), the sonogram display will not be useful (all the data at the high or low end of the colour map). To set an appropriate scale value, open the Channel Information dialog for the RealWave channel and click Rescale. This adjusts the scaling so that the data maps reasonably into integer range.

Top db and range

In most cases, the useful dynamic range of the signal (the ratio between the largest and smallest data) is less than 96 dB and you may prefer to have a smaller range displayed in the sonogram. You will probably want to set the Top dB field so that the biggest signal in your data is represented. For a waveform channel (or a RealWave channel after a Rescale), a top level of 91 dB is probably sufficient for all cases; a smaller value may be appropriate if the data does not reach full scale.

The bottom end of the displayed dB range is determined by Top db - Range dB. The smaller you set the range, the more detail you will see, but signals below the bottom of the range will all be off the bottom of the scale.

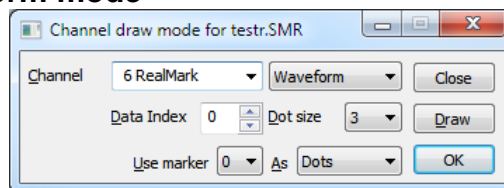
Waveform SkyLine WaveMark and Cubic spline

These modes apply to waveform, RealMark and WaveMark channels. Waveform mode joins the data points with straight lines. Skyline joins points with horizontal and vertical lines (not WaveMark channels). Cubic spline mode joins the points with smooth cubic curves based on the assumption that the first and second derivatives of the data are continuous at the data points. Cubic spline mode becomes waveform mode in Windows metafile output.

WaveMark is for WaveMark channels only and is the same as waveform mode but also draws the selected marker code.

Extra fields for RealMark channels in Waveform mode

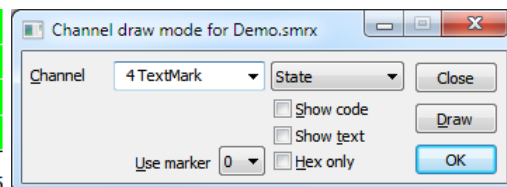
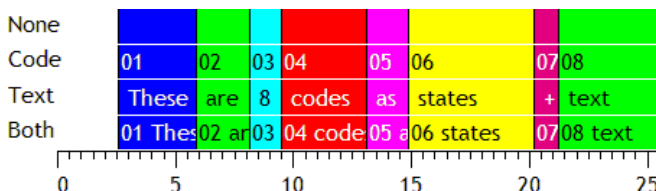
RealMark channels have extra fields in Waveform mode. A RealMark channel can have multiple data values attached to each point; Data Index selects the value to display (indices start at 0). The As field specifies how to connect the waveform points. If Dots is selected, the Dot size field can be used. The other As options are Line, Default (same as Line) and Skyline (which is the same as selecting Skyline as the draw mode).



Set WaveMark codes

If the channel holds WaveMark data you can identify events that cross a line. Hold down Alt+Ctrl and click and drag a line over the events you want to identify. On mouse up, a dialog opens in which you can set codes for the intersected events. If the channel holds multiple traces, all traces are used to select events.

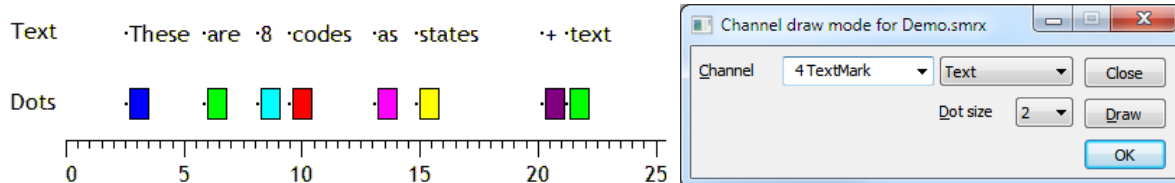
State



This drawing mode can be applied to a marker, TextMark, RealMark or WaveMark channel. The state is set by the selected marker code and persists up to the next marker on the channel. States are drawn in the same colours as set for the WaveMark codes except that code 00 is drawn in the channel background colour. The initial state at the start of the file and before any markers are found is assumed to be 00. You can choose to display the marker code and if this is a TextMark channel, the text.

If you set a marker filter on a channel drawn in state mode, this extends states through the filtered out markers, which may not be desirable. If a channel is in this mode, the Cursor Values dialog reports the displayed state.

Text



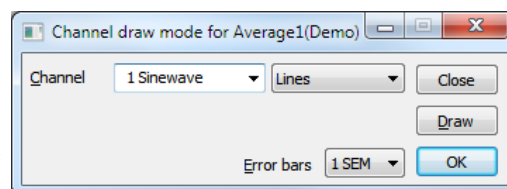
This display mode applies only to TextMark channels. Each item draws as a dot at the TextMark time followed by the text. You can override the dot and text colours with the channel primary and secondary colours in the colour palette. In Text mode the Marker code is not displayed and has no effect on the displayed text.

Result view drawing modes

There are seven drawing modes for result views: Histogram, Line, Dots, SkyLine, Cubic Spline, Raster and Raster lines. The last two can be used if you checked the Raster data box when you created the result view. In Dots mode you can also choose to display large dots.

Error bars

If your result view has associated error information, for example a waveform average with error bars enabled, you have an extra control. You can choose from None, 1 SEM, 2 SEM or SD. It should be emphasised that error bars only have meaning if the data points that contribute to the average have a normal distribution about the mean. Given this, then 1 SEM shows ± 1 standard error of the mean, 2 SEM is ± 2 standard errors of the mean and SD is ± 1 standard deviation.



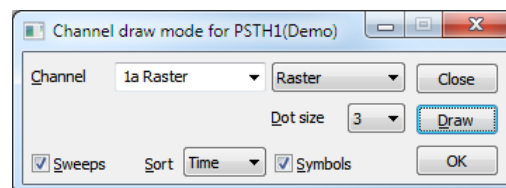
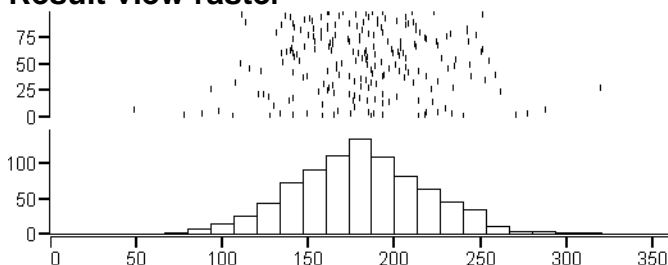
If each point of your data can be modelled as a constant "real" value to which is added normally distributed noise with zero mean, then you would expect the measured mean value to lie within 1 standard error of the mean (SEM) 68% of the time, or within 2 SEM 95% of the time. The standard deviation represents the width of the normal distribution of the underlying data at each data point.

DotSize

Dot size

If the display will show the data as dots, the Dot size field appears and you can set any dot size from 0 (the smallest mark possible) to 1 to 9 times the thickness of the lines set for drawing data in the Display tab of the Edit menu Preferences. If you set a size to match or exceed the Round dot minimum size field in Edit menu Preferences, the dots will be drawn as circles. Round dots take longer to draw than square dots.

Result view raster



If a result view channel has associated raster data, there are two more drawing modes: Raster and Raster lines. Raster mode shows each event as a dot, Raster lines mode shows each event as a short vertical line. The Dot size field controls the dot size and vertical line length. In Raster lines mode you can check the Centre line box for a horizontal line through each sweep.

Extra features with auxiliary values

If you set an auxiliary channel in the peri-stimulus, event correlation or phase histograms setup dialog or set auxiliary values from the script language, you can choose the sweep sort order. The **Sort** field can be set to **Time** or **Sort 1** to **Sort 4**. **Time** presents the sweeps in the order that they were recorded. **Sort n** sorts the sweeps based on the sort value n. Sort value 1 is set if you set an auxiliary channel in the result view setup dialog. The script language can set all the sort values with the `RasterSort()` command.

If you check **Sweeps**, the y axis is a sweep count and all sweeps are evenly spaced in the y direction, otherwise it is the value of the item selected by the **Sort** field and the sweeps are spaced out based on the value set in the **Sort** field. If you sort by **Time**, the sweep number is the order in which sweeps of data were added into the result view.

Each raster can store 8 times and display them as symbols (circle, cross, square, up triangle, plus, diamond, down triangle, filled square) if the times lie within the x axis range. Symbol time 1 (circle) is set if you select an event channel as the auxiliary channel in the result view setup dialog. Use `RasterSymbol()` from the script language.

XY Draw Mode

This command sets the drawing style of XY window data channels. **OK** makes changes and closes the dialog, **Apply** makes changes without closing the dialog. The **Cancel** button closes the window and ignores any changes made since the last **Apply**. The **Channel** field sets the channel to edit, or you can select **All channels**. If you change channel, the dialog remembers any alterations so there is no need to use the **Apply** button before changing channel unless you want an immediate update.

Join style

Style	Effect
Not Joined	No lines drawn between data points.
Joined	Each point is linked to the the next by a straight line, drawn in the channel primary colour.
Looped	Like Joined except that the last point is linked to the first.
Filled	The points are not joined, but the shape made by joining is filled with the XY channel fill colour (the channel secondary colour).
Fill and Frame	Equivalent to Filled followed by Looped.
Histogram	The points are treated as a histogram. Bins are defined by each consecutive pair of points. The start and end of each bin are set by the x co-ordinates of consecutive points. The bin amplitude is set by the first point. You need one more data point than there are bins to define the final bin width. Although it is normal for data points to be set in order from left to right with equal width bins, this is not a requirement.

The line and fill colour is set by the **Change Colours** dialog.

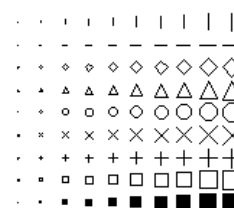
Line type and Width

These two fields set the type of line joining data points. The **Width** field determines how wide the line is in units of half the data line width set by the **Edit menu Preferences** dialog. Set 2 for the normal line width, 1 for half this width. If the width in pixels is greater than 1, the **Line type** field is ignored and the line is drawn as a solid line.



Marker and Size

The **Size** field sets the marker size in units of Points (a point is 1/72nd of an inch or approximately one screen pixel). A size of 0 makes the markers invisible. You can set sizes up to 100 points. There is a wide range of marker styles to choose from. The picture shows a screen dump of all the marker styles in sizes 1 to 10. If you need to tell them apart on screen, sizes below 3 should be avoided. If you have excellent eyesight and a high-resolution printer, size 1 is viable in printed output. The **Marker** and **Size** settings have no effect in histogram mode as markers are not drawn.



Interrupting drawing

Despite our best efforts to draw huge data files quickly, drawing can take a long time. You can interrupt it with the `Ctrl+Break` key combination. The `Break` key is usually at the top right of the keyboard and is often labelled `Pause` with `Break` on the front. If your system has sound enabled and you have selected a sound for “Exclamation”, Spike2 plays this sound to confirm that drawing has been abandoned for the current channel. Screen areas that have not been filled with data are filled with a uniform background colour.

Spike2 can detect that a drawing operation is likely to be slow. If it is, the mouse pointer changes to the hourglass cursor to indicate that you have to wait. If a drawing operation takes more than 1 second, the window title changes to remind you that you can use `Ctrl+Break` to interrupt the drawing.

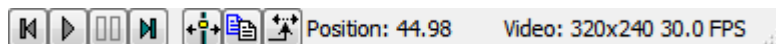
Spike2 may interrupt drawing during data sampling if the host computer is not keeping up with the data capture. This is unusual, and normally only happens when you run at high sample rates with a slow interface (such as USB 1). It can also happen with a severely fragmented disk system that has become very slow, or if you have very little free memory and a lot of paging to disk is taking place.

If a drawing operation to a time view takes more than 2 seconds, Spike2 will try to break it up into multiple drawing operations as long as the channels that require painting are not in Overdraw WM mode, are not overdrawn and are not in triggered overdraw mode. If this is not possible, after 5 seconds of uninterrupted drawing, the operating system may mark Spike2 as “Not Responding”. This can lead to problems, as once the drawing operation ends, the system may decide that the invalid area that was being redraw still needs to be redrawn, which restart the process.

Multimedia files

This View menu command opens multimedia files associated with the current time view; it is not enabled unless there are files with names matching the data file. If your data file is called `fName.smr` or `fName.smr`, the associated multimedia files are in the same folder and have names of the form: `fName-N.avi`, where the `N` stands for 1 to 4. Files do not open if they use an unrecognised format, are corrupt, use a video or audio codec that is not installed or if your computer is not equipped to replay the file.

If a multimedia file contains only audio, it folds up the video display area and you can only size the window horizontally. The multimedia window displays images as close to the time of the right-hand edge of the time view as it can. If you scroll the time view, the multimedia window scrolls to match. If you use the View menu Rerun command, any multimedia windows track the rerun and replay any audio through your sound card.



The control bar at the bottom of each multimedia window seven buttons followed by the current multimedia file position in seconds and information about the multimedia file. If you hover the mouse pointer over the buttons tool-tips appear to remind you of the button function.

The first four buttons control replay. The arrow button replays the file from the current position to the end without opening the View menu Rerun dialog; the next button stops playing. If the multimedia file contains video, the first and last button will attempt to step the video to the previous and next frame in the AVI file, otherwise they are hidden.

The fifth button links the displayed multimedia position to cursor 0 in the associated time view. Cursor 0 is made visible, if it is hidden. If you click this button, any replay will be cancelled. Likewise, the replay button cancels the link between cursor 0 and the multimedia window. When the link is made, dragging cursor 0 in the time view will also change the current position in the multimedia window. Likewise, using the step buttons will also move cursor 0 in the time view.

The sixth button copies the video image at the current multimedia file position to the clipboard. This button is hidden if there is no video in the file. The image is copied at the native resolution of the multimedia file, not at the current screen size.

The right-hand button opens a dialog in which you can set the time offset between the multimedia time and Spike2 time. This button is hidden if the file does not contain video data.

Script users have additional controls over the multimedia window. For example, they can replay individual multimedia windows independently of the associated time view. They can also grab the video image in a variety of formats (RGB, monochrome, HLS) for further processing. See the `MMOpen()` script command.

A multimedia file contains audio and/or video information. In most cases, the files will have been recorded by the s2video application. However, there is nothing to stop you renaming any avi file to match your data document. [Click here for more information on system requirements.](#)

Multimedia use makes heavy demands on the hardware of your computer. The more powerful your hardware, the better the result you will get.

Multimedia Offset

This dialog lets you view and change the time offset in the Spike2 file that corresponds with the start of the multimedia data. Ideally, the offset will be 0. However, it is very common for there to be a significant fraction of a second time offset (in either direction) between Spike2 time and the multimedia time. Negative offsets move the video data earlier (in Spike2 terms), positive offsets move it later.

We attempt to store the delay in the source .avi file so that you only need set this value once per data file. The s2video application can be configured to set an offset if your hardware generally has a constant delay.

Alignment of video

An AVI file holds a list of video frames (some of which may have no data if the frame was dropped), and the only timing information is the frame rate, which is specified as the ratio of two 32-bit numbers. The only controls we have to line this up with the Spike2 time is by adjusting the frame rate and by adjusting the offset of the first frame. There is an option in s2video to calculate the frame rate based on number of frames (and skipped frames) in the AVI file divided by the elapsed time (as seen by Spike2).

As long as the video recording process was not stymied by frames being lost due to insufficient bandwidth in the system, and the frame rate stays constant, it should be possible to align the video with the Spike2 data.

Spike Monitor

The View menu Spike Monitor command opens a new window that displays all the WaveMark channels in the current time view in a grid. The command is disabled if there are no WaveMark channels in the current time view. See the Spike Sorting chapter for more details of the Spike2 Monitor.

Font

You can select the font that is used for each window in Spike2. In time, result or XY views the font size changes the space allocated to data channels. Smaller fonts give more space to the channels, however fonts need to be large enough to read! The data view dialog is a standard operating system font dialog. In a text-based view, this command opens the editor settings dialog where you can set fonts for all the text styles supported by the view type. This is described in the Edit menu Preferences under the General tab.

There are other places in Spike2 where you can set fonts for specific items, for instance the Vertical markers dialog and the Printed Header and Footer dialog. These open standard operating system font dialogs.

ClearType fonts

If you are working with a flat panel display, you should check that you have enabled smoothing the edges of fonts and have ClearType enabled as this can greatly enhance the appearance of text, especially small font sizes. In Windows XP, font smoothing is enabled in My Computer->Properties->Advanced->Settings->Visual Effects and make sure that Smooth edges of screen fonts is checked. In Windows XP, ClearType is enabled by right clicking on the desktop and choosing Properties->Appearance->Effects... then check the box Use the following method to smooth edges of screen fonts and select ClearType.

Dialog font

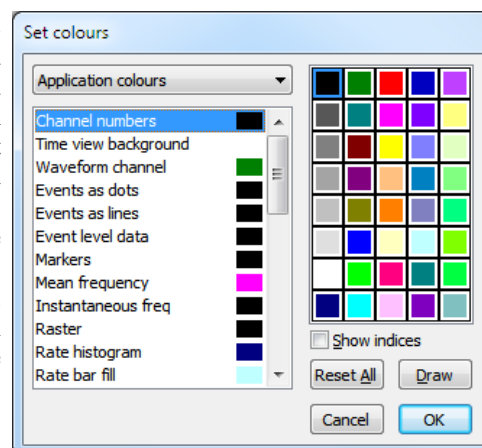
The size of the font used for all Spike2 dialogs is fixed (but see the `DlgFont()` script command). At the current time, the only way to change the size is to change the Windows system display settings to alter the DPI value (Dots Per Inch). This will affect all programs, not just Spike2.

Use Colour and Use Black And White

If you have a monochrome monitor or if you have to use a printer which does not handle colour well, you can choose to display your data files in black and white. The **Use Black And White** menu item switches from colour to black and white displays. If you change to monochrome, the menu item changes to **Use Colour**. This option saves you from the tedious task of changing every colour in the colour palette manually.

Change Colours

The **Change Colours** dialog controls time, result and XY views display colours (the **Font** dialog controls colours in text-based views). To open the **Change Colour** dialog, use the **View** menu **Change Colours** command or click on the palette in the main toolbar. You can choose the colours that are used for almost everything in Spike2 (sonogram colours are changed by a different dialog). If you open the dialog with an active time, result or XY view, the dialog has multiple pages. Select a page with the drop-list at the top left. Pages are: **Application colours**, **Marker colours**, **View colours**, **Channel primary colour**, **Channel secondary colour**, **Channel background colour**. The **Application** and **Marker** colour pages are always available, the remainder are available when a time, result or XY view is active.



Application colours dialog

Application colours

To change colours, select one or more items in the list on the left, and then click a colour in the palette on the right. You can check the result of your action with the **Draw** button. **Cancel** removes the window and undoes any changes. **OK** accepts changes and closes the dialog. The **Reset All** button returns the list and the palette to a standard set of colours. You can set the following (bracketed text means it can be overridden):

Channel numbers	The channel number, used to identify and select data channels and drag them to new positions.
Time view background	Background colour of the entire time view (view background)
Waveform channel	Waveform data in time views (primary). Also used for waveforms in the spike shape and digital filter dialogs.
Events as dots	Event data drawn as dots (primary).
Events as lines	Event data drawn as vertical lines (primary) and horizontal line (secondary).
Event level data	Level event data (primary).
Markers	Marker codes and associated dot, plus RealMark waveform (primary).
Mean frequency	Events draw as mean frequency (primary).
Instantaneous freq	Events drawn as instantaneous frequency (primary).
Raster	Time view and Result view raster responses (primary), Time view trigger event and Result raster centre line (secondary).
Rate histogram	Border colour for events drawn as a rate histogram (primary).
Rate bar fill	Fill colour for events drawn as a rate histogram (secondary).
Result background	Result view background (view background).
Result lines	Result view data as lines (primary).
Result dots	Result view data as dots (primary).
Result skyline	Result view data as a skyline(primary).
Result histogram	Border of result view data as a histogram (primary).
Result bar fill	Fill of result view data as histogram (secondary).
TextMark text	TextMark data drawn in Text mode
Cursors	The colour to use for horizontal and vertical cursors and cursor labels.
Controls	This is currently unused.
Grid	All axis grids.
Axes	All axis lines, ticks and labels.
XY view background	The XY view background colour (view background).

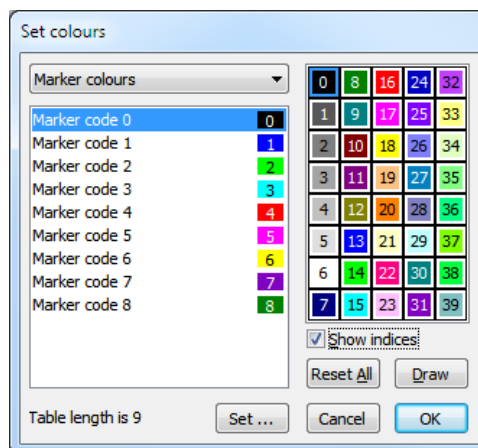
Not saving to disk	Used to draw data displayed in a sampling window that is not being written to disk. This colour is only used when the most recent data is drawn at the right hand edge of the window (when the scroll bar thumb is at the far right of the time view). Also used for non-triggered data in the spike shape dialog. Set this colour the same as Time view background to use the normal colours.
Errors (SD/SEM)	All result view error bars and envelopes (secondary).
Curve fit	Fitted curves.
Cluster background	The background colour for the clustering dialog.
WaveMark background	The background colour for all the spike shape dialogs.
Vertical markers	Vertical marker colours if marker colours are not used.
XYChannel data	XY view data channels (primary).
XY Channel fill	XY view fill colour (secondary).
XY key	XY view key

If you set a channel or text colour that is too similar to the background, Spike2 will change the colour to keep the item visible. There is an option to defeat this in the Edit menu Display Preferences. Script users can set application colours with the `ColourSet()` command.

Marker colours

Several drawing modes draw Markers and marker-derived data (TextMark, RealMark, WaveMark channels) in different colours, depending on a marker code. Marker codes can take values from 0 to 255. Prior to version 7.07, there were 9 marker colours, numbered 0 to 8. Marker code 00 always uses marker colour 0. All other marker codes used colour $(c \bmod 8) + 1$, where c is the marker code and \bmod is the modulus (remainder) operator. Marker codes 1 to 8 used marker colours 1 to 8, marker codes 9 through 16 also used marker colours 1 to 8, and so on.

Click **Set...** to change the number of marker colours in the range 3 to 255. With m colours, marker code c is assigned marker colour $(c \bmod (m-1)) + 1$. If you increase the marker colours, new colours start set to black. Script users can set marker colours with `ColourSet()`. For WaveMark data, the colour is for the first trace, the rest fade to the background.



Marker colours dialog

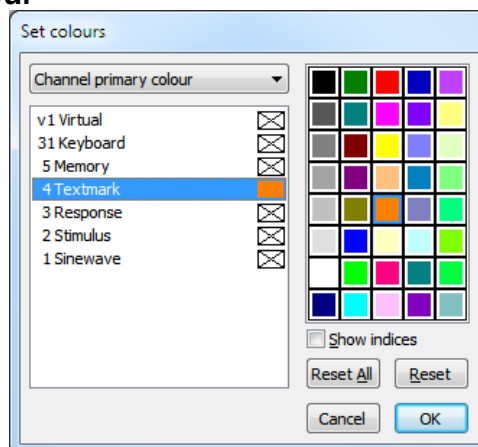
Show indices

Check this box to display the palette number for each colour in the palette and the item number (where appropriate) in the list to the left. This is a convenience for script programmers.

Channel primary, secondary, background colour

These three pages assign colours to data channels in the current time, result or XY view and override any application colour. The primary colour is used as the drawing colour for lines, waveforms, events, and histogram outlines. The secondary colour is for filling histograms and XY channels, the raster display trigger point, horizontal centre lines for events and result view raster line mode, the text colour in Text drawing mode and for drawing the SEM and SD in result views. The channel background colour overrides the view background for the area occupied by the channel data. An X in a box marks a channel with no colour override.

Changes made on this page are applied immediately, so there is no **Draw** button. You can **Reset** the selected channels back to the standard colours set in the Application colours page. Channel colour overrides are stored in the resource file associated with the data file and in the sampling configuration file for new data files. Script users can set channel colours with the `ChanColourSet()` command.



Channel primary colour dialog

View colours

This page allows you to override the application colours set for the current view. At the moment you can only override the view background colour. The equivalent script command for this is `ViewColourSet()`. View colour overrides are stored in the resource file associated with the data file and in the sampling configuration file for new data files.

Assign a colour

There are two ways to assign a colour to the selected item in the list.

1. Click on the palette to the right to assign a palette colour to the list item.
2. Double-click the list item to open a colour picker to choose a colour.

The Reset button restores the selected item to its default state (if there is one) or cancels a colour override.

Changing the Palette

To change a palette colour, double click it to open the colour picker and select a replacement colour. The first seven palette colours form a grey scale from black to white and cannot be changed. You can replace the palette colour with any standard colour, or you can click the **Define Custom Colours...** button to select an arbitrary colour. Click **OK** or **Cancel** to exit.

The colour palette is owned by the Spike2 application, not by the data file and is stored in the registry and also with the Spike2 configuration in `default.s2c`. When Spike2 starts, the palette and Application and Marker colours are read from the registry; if there are no colours to read, the colours in the configuration file are used.

Script language

The script language function `ColourSet()` controls the palette, Application and Marker colours. The `ChanColourSet()` function controls the Channel primary, secondary and background colours and The `ViewColourSet()` command controls the view colours.

Changes at version 7.07

Prior to version 7.07, the application, view and channel colours were all controlled by the colour palette. All colourable items had an index into the palette and were displayed with the colour at that index. This made sense when Spike2 was originally written as most systems had 16 colours, some had 4 and some had only 2. It was possible to generate intermediate colours by mixing dots of different hues, but this led to unpleasant screen effects. Apart from limiting the colour choice to the colours in the palette, using palette indices had the additional effect that a change to a palette colour might change items other than those that were intended.

From version 7.07, all items have a free choice of colour, defined by RGB (Red, Green and Blue) values in the range 0 to 1. For example, a RGB value of (1,0,0) is red, (0,0,0) is black, (1, 1, 1) is white and 0.5, 0.5, 0.5 displays as a mid grey. We have preserved the palette as it is useful to have a set of colours that are easy to apply. A click in the palette sets the RGB value of the palette colour; the palette index is of no consequence.

This change is transparent to most users. The only difference is that if you double click in the palette to change the colour, the change applies to the currently selected item in the list on the left of the dialog. Previously, if you changed palette item *n*, this would change the colour of any Application, View or Channel colour that happened to be pointing at palette index *n*.

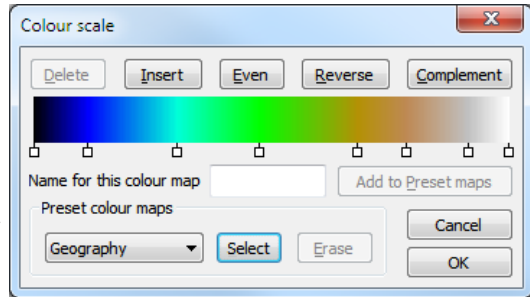
The changes should make it much easier to set colours using a script. However, we have preserved the old script commands so that old scripts will continue to work. We very much encourage you to avoid the old script commands in new scripts (unless they must also run in older versions of Spike2). The new script commands that allow you to set and read back colours using RGB values are: `ColourGet()`, `ColourSet()`, `ChanColourGet()`, `ChanColourSet()`, `ViewColourGet()` and `ViewColourSet()`. You can still use the following commands, but we suggest that you avoid them in new code: `Colour()`, `ChanColour()`, `ViewColour()`, `XYColour()`, `PaletteGet()` and `PaletteSet()`.

We have also separated out the Marker colours (previously called WaveMark colours). This also allows us to provide more marker colours, if needed.

The new system is more open-ended than the old, and allows us to add further colours and colour overrides in the future.

Sonogram Colours

The Colour Scale dialog can be opened by the Sonogram Colours command in the View menu and from the clustering dialog View menu Density Colour Map command. You can also open this dialog by right-clicking on a time view sonogram display and selecting Sonogram Colours from the context menu. The dialog sets the colour gradient for the time view sonogram display mode and for the clustering density plots. You can choose from a range of built-in and user-defined colour maps and create new colour maps. Apart from the standard OK and Cancel buttons, the dialog is in three sections: an upper part where you create new maps, the centre line, where you name and save created maps, and the Preset colour maps section where you select and manage named maps. The same dialog is also used for the cluster density colour scale.



The colour map

The small squares below the colour map mark fixed colour points. Single click a small square to select it. Double-click a small square to change its colour. The colours between the fixed colour points are formed by linear interpolation. Click and drag the squares sideways to change their position. You cannot move the first and last fixed points. You can add a new fixed point by double-clicking on the colour scale away from any small square. The buttons above the colour scale are:

- | | |
|-------------------|--|
| Delete | This deletes the selected fixed point and is disabled if there is no selection. |
| Insert | If there is a selected fixed point, this adds a new point next to the selected point on the side with the larger gap. If there is no selected point, this adds a new fixed point in the largest gap between existing fixed points. You can also add a fixed point by double-clicking the colour map clear of any fixed point. You can have a maximum of 10 fixed points. |
| Even | This spaces the fixed points evenly across the colour bar. |
| Reverse | This reverses the order of colours. Reversing twice restores the original. |
| Complement | This replaces each colour by its complement, formed by subtracting each colour from white. Complementing twice restores the original. |

Name for this colour map

You can save up to 10 user-defined maps in the registry. Set a name and click the Add to Preset maps button to add a new colour map to the list. If the list is full you must use the Erase button to remove a colour map before adding a new one. If you type a name that cannot be used, the Add to Preset maps button is hidden and a text message explaining the problem replaces it. Names of user-defined maps must be unique and may not include the vertical bar character.

The names are stored in the registry in the HKEY_CURRENT_USER\Software\CED\Spike2\Settings\ColourMap folder with the names ColourMap0 to ColourMap19. The map used for drawing sonograms is stored in ColourMap0. This map is loaded each time Spike2 starts. The cluster density map is stored in ColourMap1. The user-defined maps are saved as ColourMap10 to ColourMap19. The names ColourMap2-9 are reserved for future maps.

Preset colour maps

The drop-down list in this section holds several built-in maps with names that suggest how they are formed, for example Rainbow, Thermal and Geography. It also holds the names of maps that you have created and saved. Click Select to copy the map with the displayed name to the displayed colour map. The Erase button can be used to delete a user-defined map from the list and the registry. You cannot delete the built-in maps.

OK

Click the OK button to accept the current map. The current map is stored in the system registry as ColourMap0 for sonograms and as ColourMap1 for clustering and is applied immediately.

Folding

This menu item is available for Output sequencer and Script views to control the code folding options. The folding options are covered in the Edit menu Preferences command in the General tab for these view types. This View menu command gives you a quick way to change the folding style and to toggle all the folds. The folding styles (illustrated for a script file) are

```
'Check view type
if ViewKind() then
  halt;
else
  Message("Time view");
endif;

'Type OK
const n% := 100, phi := 0.01;
var i%, x[n%], y[n%];
for i% := 0 to n%-1 do
  x[i%] := Sin(i%*phi);
  y[i%] := Cos(i%*phi);
next;
```

Example of folding in a script

Arrows, Plus and Minus, Circles and Lines, Squares and Lines:

<pre>'Check view type if ViewKind() then halt; else Message("Time view"); endif; 'Type OK const n% := 100, phi := 0.01; var i%, x[n%], y[n%]; for i% := 0 to n%-1 do x[i%] := Sin(i%*phi); y[i%] := Cos(i%*phi); next;</pre>	+	<pre>'Check view type if ViewKind() then halt; else Message("Time view"); endif; 'Type OK const n% := 100, phi := 0.01; var i%, x[n%], y[n%]; for i% := 0 to n%-1 do x[i%] := Sin(i%*phi); y[i%] := Cos(i%*phi); next;</pre>	⊕	<pre>'Check view type if ViewKind() then halt; else Message("Time view"); endif; 'Type OK const n% := 100, phi := 0.01; var i%, x[n%], y[n%]; for i% := 0 to n%-1 do x[i%] := Sin(i%*phi); y[i%] := Cos(i%*phi); next;</pre>	⊞	<pre>'Check view type if ViewKind() then halt; else Message("Time view"); endif; 'Type OK const n% := 100, phi := 0.01; var i%, x[n%], y[n%]; for i% := 0 to n%-1 do x[i%] := Sin(i%*phi); y[i%] := Cos(i%*phi); next;</pre>
---	---	---	---	---	---	---

The four folding styles

The remaining folding commands are:

Toggle all folds

This searches the text from the start to find the first fold, then sets all the folds in the file to the opposite state. The short-cut is **Ctrl+Shift+T**. To toggle an individual fold, click on the fold mark.

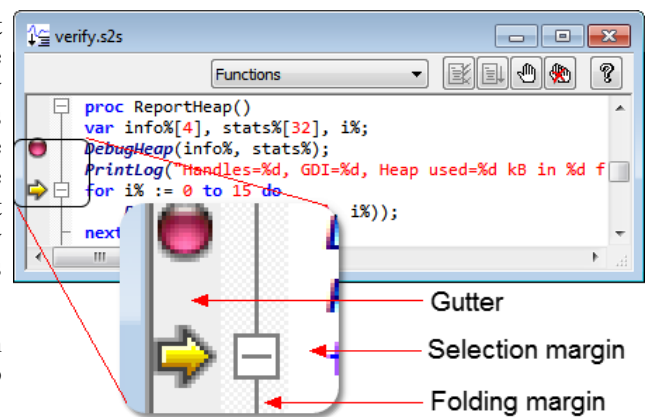
No folding

This hides the fold margin. If this option is selected and text is folded, it is unfolded.

Show Gutter

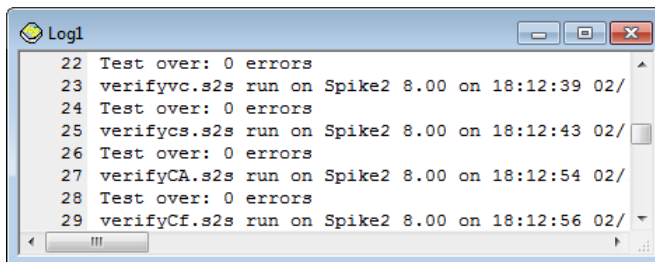
The gutter is an area to the left of the text in a text view, usually with a grey background, that can be used to select lines of text, except in a script view where clicking on it sets break points. The gutter is also used in script views to display the current line pointer. This menu command shows and hides the gutter, equivalent to the `Gutter()` script command. The gutter shares the same display background colour as line numbers. The gutter is normally visible.

If you turn the gutter off, there is a selection margin to the left of the text that can be used to select lines of text by clicking and dragging.



Show Line Numbers

You can choose to display line numbers in any text view. Line numbers usually have space for up to 5 digits, but for more than 99,999 lines of text you can use the `ViewLineNumbers()` script command to increase the displayed digits. This menu command makes the line number area visible if it is invisible and hides it if it is visible. You can change the appearance of the line number region with the View menu Font command. Select the Line number style, and a line number will appear in the example window so you can preview any changes.

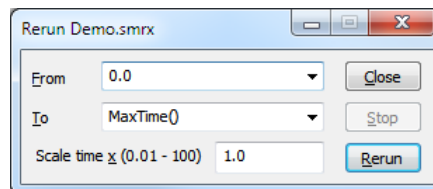


Log view with line numbers on and the gutter off

With line numbers on, you will usually want the gutter off, except in a script where the gutter is used for break points and to mark the current line. You can use the line number margin to select complete lines of text.

ReRun

This option can be used in a Time window to simulate data sampling for demonstration and training purposes. From and To define the time range to use. Scale time sets the number of seconds of data to rerun every second. Rerun starts the simulation and Stop cancels it. Unlike real sampling, there is no sample control window and the output sequencer and on-line waveform replay are not available.



When the update point reaches the right-hand window edge, the window scrolls to keep the update point at the right-hand edge. You can use the scroll bar to move through the document, but only up to the current end. If you move the scroll bar fully to the right, the window will scroll automatically, if the scroll bar is not at the right, the window will not scroll, but the scroll bar position will change as time passes to show its relative position in the document. The document will keep running until it reaches the end or you use the Stop button.

Any open multimedia windows associated with the time window also rerun and replay the audio signal to stay aligned with the current replay point.

In place of scrolled displays, you can have a paged update. Open the View menu Display Trigger dialog, select None as the trigger channel, check Enable trigger and click OK.

Link to Offline waveform output

To rerun to match a waveform playing through the 1401 DACs or through a sound card in your computer, check the Rerun the file to Match the waveform output box in the Offline waveform output dialog.

Annotate

This menu item is visible if the current view is a script and is enabled if the script is compiled. Select the option to display the compiled script code below each script line. Select it again to hide the compiled code. We use this annotated view to diagnose script compiler problems.

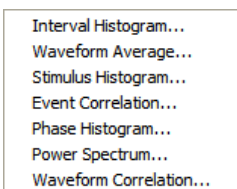
9: Analysis menu

Analysis menu

This menu creates result and XY windows that hold analysed data from time view channels, manages the memory buffers, deletes, duplicates and calibrates channels, gives access to the marker filter control and manages spike shapes. It also holds the digital filtering command.

New Result View

This command is available when the current window is a time view. It is a pop-up menu from which you can select an analysis type. This leads into a dialog where you define the parameters to construct a result window.



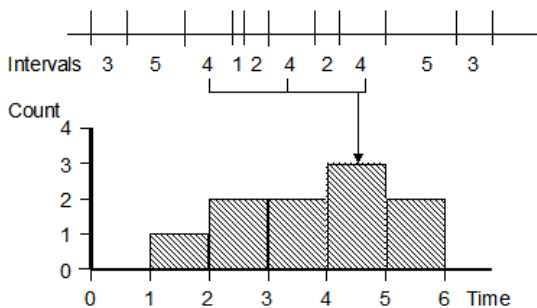
A result window holds arrays of data that can be drawn as histograms or waveforms. There is one data array for each channel you analyse. Some types of analysis can store additional data, for example the peri-stimulus histogram can store event times for a raster display and the waveform average can store extra data so that you can display error bars in the result.

Once you have set the required values in the dialog box, Spike2 creates a result window with all data values set to zero. A new dialog appears to prompt you to select a region of the original time window to analyse. The results of analysing different areas can be summed.

The following analyses are implemented: interval histogram, waveform average, peri-stimulus time histogram, event time cross correlation, event phase histogram, power spectrum and waveform correlation.

Interval histogram

An interval histogram (INTH) displays the frequency histogram of the intervals between events on a selected channel. All times in a Spike2 data document are expressed as a multiple of the basic time unit set when the data document was created. Therefore you cannot get better time resolution than the basic time unit (usually in the range 1 to 50 us).

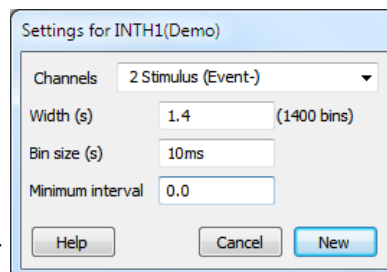


The picture illustrates how an interval histogram is formed. To make the diagram easy to understand, we have shown several events with all the intervals between them as a whole number of seconds. The interval histogram formed from these events has bins set with a width of one second.

The interval between each pair of events is divided by the bin width to give the bin number to increment (fractional bin numbers are rounded down, the first bin is bin 0). In this case, the bin width is 1, so there is no rounding. In a more realistic case with an interval of 2.3 milliseconds and a bin

width of 1 millisecond, bin number 2 (spanning the time period 2 to 3 milliseconds) would be incremented.

Event times are rounded down to the nearest base time unit, so an event time of τ base units means that the actual time was greater than or equal to τ and less than $\tau+1$ units. An interval of n units means that the real time interval between two events was greater than $n-1$ units and less than $n+1$ units. If you need to form interval histograms of very short periods, make sure that you have set the duration of the base time unit short enough to resolve the information you require.



The Channel field selects one or more event channels; type a list of channels or select from the drop down list. Each channel generates a histogram in the result view. If you type a channel list or use the selected channels, the order of the channels in the result view is the order of channels as you enter them.

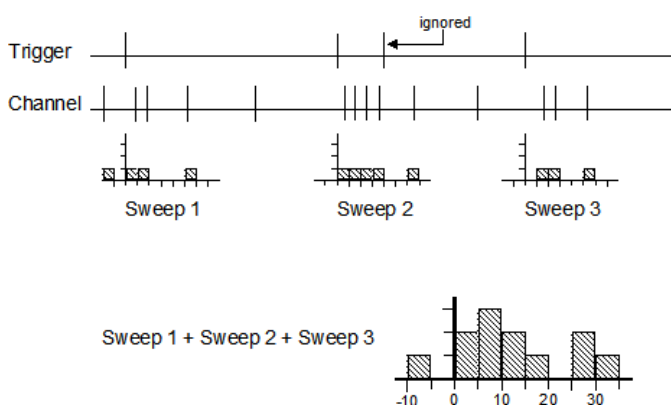
Width sets the time width of the histogram from left to right. The Minimum interval field is the smallest interval to put in the histogram and sets the left hand end of the x axis. It must be positive and is usually zero. Bin size is rounded to the nearest multiple of the underlying time units used in the time window. The number

of bins (Width/Bin size) must be at least 1. The maximum size is limited only by available system memory. You can enter the Width, Bin size and Minimum interval fields in milliseconds by following the numeric value with `ms`. You can force microseconds with `us`. For example, `0.001`, `1ms` and `1000us` are all the same value.

The **New** button accepts the values in the dialog and creates a new result window. This dialog can also be activated by the **Process Settings** menu command, in which case the **New** button is replaced by a **Change** button. **Change** clears any previous results.

Click the **New** or **Change** button to open the **Process** dialog in which you set the time range of data to analyse. Only events that fall within the time range are considered, intervals that start or end outside the time range are ignored.

Peri stimulus time histogram



A peri-stimulus time histogram (PSTH) forms a histogram of events on one or more response channels around a stimulus event on a trigger channel. Each source channel generates a separate histogram.

Each trigger event that does not lie within the previous sweep generates a new sweep. Trigger events that fall in the previous sweep are ignored. If you require overlapped sweeps, use the event correlation analysis. Sweeps are accumulated to produce a histogram. You can scale the result into spikes per second rather than event counts.

In the Settings dialog, the **Channel** field selects the channels to analyse. The **Width** field sets the width of the histogram in seconds and the **Bin size** field sets the width of each bin. Their ratio sets the number of bins in the histogram. The **Offset** field sets the pre-trigger time to display. If the time offset is 0.5, the histogram time axis starts at 0.5 seconds. All times are rounded to the nearest multiple of the basic time unit. You can supply values for the **Width**, **Bin size** and **Offset** fields in milliseconds by appending `ms` to the value or in microseconds by appending `us`, for example `0.01` seconds can be entered as `10ms` or as `10000us`.

The **Trigger** field sets a channel to use as the trigger, or you can select no trigger. If you do not select a trigger channel, the process command uses the start time as the trigger time for a single sweep. If you select a trigger channel, the process command defines a time range to search for triggers.

Spikes per second

If you check the **Spikes per Second** box, the histogram shows equivalent spike rate rather than the event count in each bin. To change from event count to rate, Spike2 divides the event count by the number of sweeps to give the spikes per sweep, and then divides the result by the bin width to give spikes per second.

You can change between count and spikes per second after you have created the result view and analysed data. Click on the result view, open the Process settings dialog and change the state of the **Spikes per second** check box, then click **Change**.

Raster data

If you check the **Raster data** box, the new result view stores the times of all the events that are added into histograms and you can draw the result as a raster display. When you create the result view, Spike2 automatically duplicates each result channel and draws the duplicated channel in raster drawing mode above the histogram. As each event uses memory and takes time to draw, the maximum number of events is limited by the memory in your system and your patience. This should not be an issue unless you work with hundreds of thousands of spikes with the result view raster enabled.

Once you have created the result window and analysed data you can delete the raster data by opening the Process settings dialog and clearing the **Raster data** check box. You cannot add raster data to an existing

result view without clearing all the histograms.

Auxiliary measurements

You can take one measurement per sweep from an auxiliary channel. If you choose a waveform or RealWave channel, the measurement is the channel value at the trigger point or 0 if there is no waveform data at the trigger. For all other channels types, the measurement is the time between the sweep trigger point and the first event on the auxiliary channel after or before the trigger. Check the **Backwards** box to search before the trigger. Pre-trigger times are negative, times after it are positive. If no event is found, 0.0 seconds and the maximum time possible in the file are used as virtual event times.

Accept sweeps

You can accept all sweeps, or only those for which the measured value lies inside or outside the values in the **Range** fields. Sweeps with no data found are treated as outside the range. When the auxiliary channel holds waveform or RealWave data the units of the range fields are the channel units, for all other types the units are seconds. When the units are seconds, you can enter values in milliseconds by appending **ms** to the value.

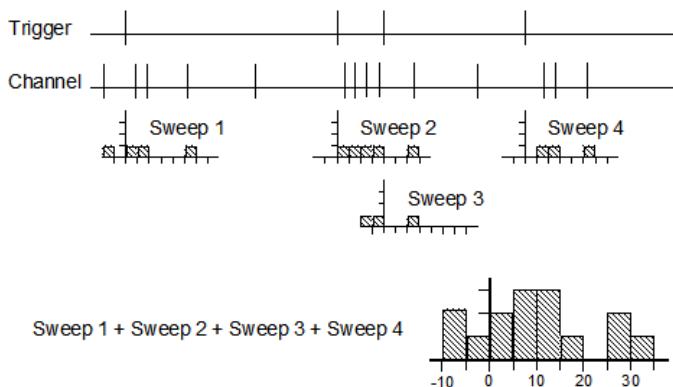
Sorting rasters

If you enable raster data, you can sort raster sweeps based on event latencies or waveform values. You set the sort mode from the Draw Mode dialog. Each raster sweep can store four sort values. The auxiliary measurements system uses the first sort value. The `RasterSort()` script command can access all the sort values.

Raster symbols

Each raster sweep can store eight event times that you can choose to display as symbols with the Draw Mode dialog. If the auxiliary measurement is an event time, and an event was found, the time is saved as the first symbol time. Script users can access all eight symbol times with the `RasterSymbol()` command.

Event correlation

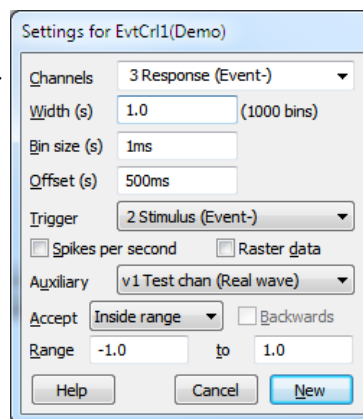


This command performs event cross-correlations and event auto-correlations between a trigger channel and one or more other event channels. A cross-correlation produces a measure of the likelihood of an event on one channel occurring at a time before or after an event on another channel. The result can be displayed as a spike rate or as a count.

Each trigger generates one sweep of analysis (unlike a PSTH where a trigger event that falls within a sweep is ignored). The zero times of each sweep are aligned, and then the sweeps are accumulated to form a histogram.

The **Channel** field sets a channel or a channel list to analyse. The **Width** field sets the width of the histogram in seconds. The **Bin size** field sets the width of each bin in seconds. The ratio of these fields sets the number of bins in the histogram. The **Offset** field sets the pre-trigger time to display in seconds, so if the time offset is **offset**, the histogram time axis starts at a time of **-offset**. All times are rounded to the nearest multiple of the basic time unit. You can set the Width, Bin size and Offset fields in milliseconds or microseconds by appending **ms** or **us** to the entered value. For example, 0.001, 1ms and 1000us all represent the same time value.

The **Trigger** field is a drop down list in which you can select a trigger channel or **Manual** mode. In **Manual** mode, the process command start time is the trigger time for one sweep. With a trigger channel, the process command sets a time range to search for trigger events.



Spikes per second

If you check the **Spikes per Second** box, the histogram shows equivalent spike rate rather than the number of events in each bin. To change from event count to rate, Spike2 divides the number of spikes in each bin by the number of sweeps to give the spikes per sweep, and then divides the result by the bin width to give spikes per second.

Raster data

If you check the **Raster data** box, the new result view stores the times of all the events that contributed to the histogram and you can use raster drawing mode for the result view channel. When you create the result view, Spike2 duplicates each channel and draws the duplicate in raster mode above the histogram.

As each event uses memory and takes time to draw, the maximum number of events is limited by the memory in your system and your patience. This should not be an issue unless you work with hundreds of thousands of spikes with raster enabled.

Auxiliary measurements

You can take one measurement per sweep from an auxiliary channel. If you choose a waveform or RealWave channel, the measurement is the channel value at the trigger point or 0 if there is no waveform data at the trigger. For all other channels types, the measurement is the time between the sweep trigger point and the first event on the auxiliary channel after or before the trigger. Check the **Backwards** box to search before the trigger. Pre-trigger times are negative, times after it are positive. If no event is found, 0.0 seconds and the maximum time possible in the file are used as virtual event times.

Accept sweeps

You can accept all sweeps, or only those for which the measured value lies inside or outside the values in the **Range** fields. Sweeps with no data found are treated as outside the range. When the auxiliary channel holds waveform or RealWave data the units of the range fields are the channel units, for all other types the units are seconds. When the units are seconds, you can enter values in milliseconds or microseconds by appending **ms** or **us** to the value.

Sorting rasters

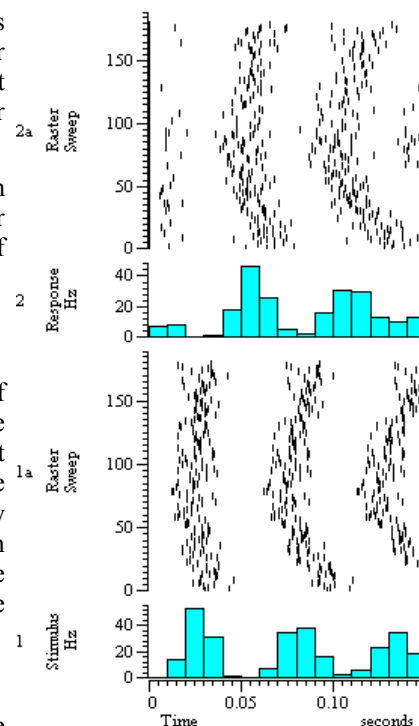
If you enable raster data, you can sort raster sweeps based on event latencies or waveform values. You set the sort mode from the Draw Mode dialog. Each raster sweep can store four sort values. The auxiliary measurements system uses the first sort value. The `RasterSort()` script command can access all the sort values.

Raster symbols

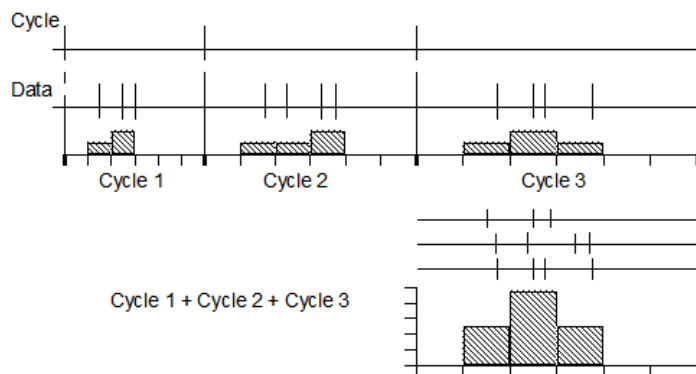
Each raster sweep can store eight event times that you can choose to display as symbols with the Draw Mode dialog. If the auxiliary measurement is an event time, and an event was found, the time is saved as the first symbol time. Script users can access all eight symbol times with the `RasterSymbol()` command.

Auto-correlation

For an auto-correlation, set the **Channel** and **Trigger** fields to the same channel. The analysis for an auto-correlation has one difference from that for a cross-correlation. In an auto-correlation, the correlation of each event with itself at time 0 in the histogram is ignored.



Phase histogram



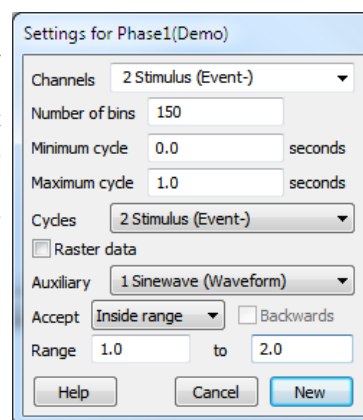
A phase histogram is used to show how events are distributed with respect to a cyclical process that may vary in cycle time. One event channel marks the start and end of cycles. The end of one cycle is the start of the next. Events on another channel are placed in bins depending upon their position within each cycle.

You can limit the range of cycle times to consider for inclusion in the histogram, either to eliminate impossible data or to show how responses vary with the cycle

times.

The **Channels** field sets the channels to analyse. The **Cycle** channel sets the channel with cycle markers. The **Number of bins** field sets the bins per cycle. The width of each bin depends on the duration of each cycle. The **Minimum cycle time** and **Maximum cycle time** fields exclude cycles that are too long or too short to belong to the data and are set in seconds. You can set these values in milliseconds or microseconds by appending **ms** or **us** to the value. Check the **Raster data** box to save the times of all events and duplicate each result channel in raster mode.

In the **Process** command, the start and end times determine the time range of the data in the **Cycle** channel to search for cycle markers. Cycles that are longer than the maximum time or shorter than the minimum time are ignored. If no **Cycle** channel is supplied, the start and end times are used as the start and end of a single cycle.



Auxiliary measurements

You can take one measurement per sweep from an auxiliary channel. If you choose a waveform or RealWave channel, the measurement is the channel value at the trigger point or 0 if there is no waveform data at the trigger. For all other channels types, the measurement is the cycle position of the first event on the auxiliary channel after or before the cycle start. Check the **Backwards** box to search before the trigger. Pre-trigger positions are negative. The cycle start position is 0 degrees, the cycle end position is 360 degrees. If an event is at time T_e and the cycle start and end times are C_s and C_e , the event position is $360 * (T_e - C_s) / (C_e - C_s)$. If no event is found, 0.0 seconds and the maximum time possible in the file are used as virtual event times.

Accept sweeps

You can accept all sweeps, or only those for which the measured value lies inside or outside the values in the **Range** fields. Sweeps with no data found are treated as outside the range. When the auxiliary channel holds waveform or RealWave data the units of the range fields are the channel units, for all other types the units are degrees.

Sorting rasters

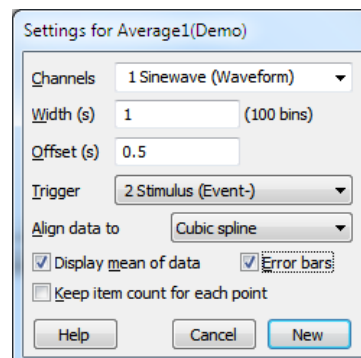
If you enable raster data, you can sort raster sweeps based on cycle positions or waveform values. You set the sort mode from the **Draw Mode** dialog. Each raster sweep can store four sort values. The auxiliary measurements system uses the first sort value. The `RasterSort()` script command can access all the sort values.

Raster symbols

Each raster sweep can store eight event times that you can choose to display as symbols with the **Draw Mode** dialog. If the auxiliary measurement is an event time, and an event was found, the time is saved as the first symbol time. Script users can access all eight symbol times with the `RasterSymbol()` command.

Waveform average

This command averages waveform channels with respect to a trigger signal, or if no trigger is set, averages waveform channels with respect to user-defined trigger times. The **Width** field sets the time range spanned by the average, in seconds. The **Offset** field sets the start time of each sweep before the trigger, in seconds. You can provide the **Width** and **Offset** values in milliseconds or microseconds by appending **ms** or **us** to the value. The **Trigger** field sets the source channel for trigger times or **Manual** trigger mode.



Display mean of data

The **Display mean of data** chooses between a display of the mean of all the sweeps or a display of the sum of all the sweeps. If you return to this dialog after forming an average you can change this setting to redraw the result without needing to process the data again.

Error bars

If you check the **Error bars** box, extra information is saved with the result so that you can display the standard deviation and standard error of the mean of the resulting data. The **Channel Draw Mode** command controls the display of the error information.

Align data to trigger

In general, trigger times will fall between samples of the waveform channel. If you are looking at features that are only a few sample points in duration, for example transient responses, the data may change significantly between the sample points. You have a choice of four methods to set the alignment of the data to the trigger.

Next data point

The data is aligned at the next sample after the trigger time. This is the method used by all Spike2 versions before 6.01.

Nearest data point

The data is aligned at the nearest sample to the trigger point. This is the best method to use if you don't want to interpolate.

Linear interpolation

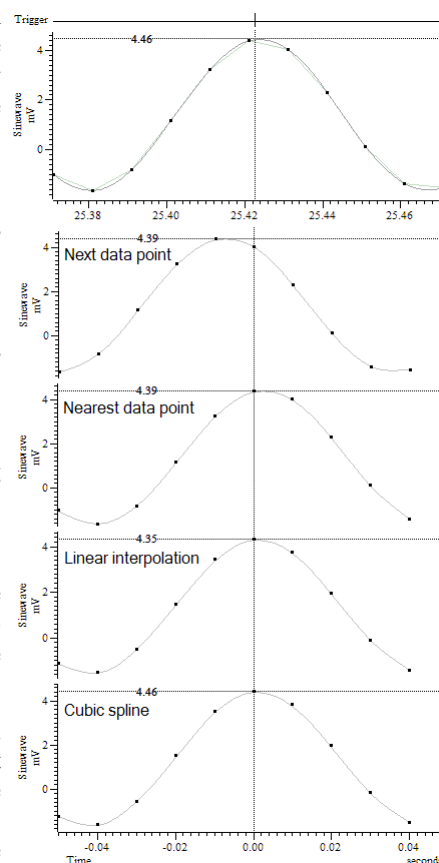
The data values are aligned to the trigger by linearly interpolating between the data points.

Cubic spline

The data values are aligned to the trigger by cubic spline interpolation. This is the best method to use if your data contains no components at frequencies above half the sample rate. It is also the slowest method.

The best method to use depends on your data. If your data is known to contain no frequencies above half the sample rate (which is what you would hope to be the case), using a cubic spline will produce the best results. If this is too slow, you could try linear interpolation, which is faster to compute, but which will tend to reduce the amplitude of narrow features.

If your data contains transients with frequency content above half the sampling rate, cubic interpolation will become inaccurate and the best you can do is choose the **Nearest data point** method. The **Next data point** method is provided for backwards compatibility with older versions of Spike2.



Keep item count for each point

If you check this box, each point in the result view keeps track of the number of items that were averaged to create it. If you do not check the box, all points are presumed to have averaged the same number of items (the sweep count). The box makes a difference when sweeps of data are truncated or interrupted by gaps or the start or end of the channel data. If you do not check the box, missing data is presumed to hold zeros, which may cause discontinuities in the average. If you check the box, missing data makes no contribution to the average. If you use this option, saved result view files will not be compatible with versions of Spike2 before 5.16 and 6.00.

New (Change) button

The New button (or Change when used from the Process Settings command) closes the dialog, creates a new result window and opens the Process dialog. With a trigger channel, the events between the start and end times set in the process dialog are triggers. In Manual mode, the start time is the trigger for a single sweep.

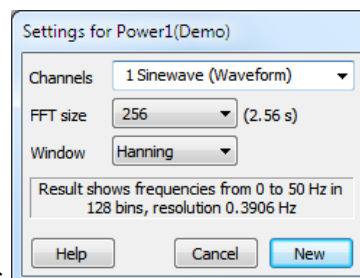
Sweep count

The result window for an average keeps track of the number of data sweeps that have been added into the average. A sweep is counted if a least one channel adds at least one data point into the average. It displays the mean waveform by accumulating the data and dividing by the number of sweeps or, if the Keep item count... box was checked, the item count for each point.

The start of the section of data to add to the average is found by subtracting the time in the Offset field from the trigger time. The data channel is then aligned to the trigger by the method set in the Align data to trigger field. If there is a gap in the waveform data, such that there is no data point that falls within time from the start of the section to the start plus the width of the average, no data is added.

Power spectrum

This command creates a result window that holds the power spectrum of a section, or sections of data. The result of the analysis is scaled to RMS power, so it can be converted to energy by multiplying by the time over which the transform was done. You can transform multiple channels, but they must have the same sample rate. Spike2 uses a Fast Fourier Transform (FFT) to convert the waveform data into a power spectrum.



The FFT is a mathematical device that transforms a block of data between a waveform and an equivalent representation as a set of cosine waves. The FFT that we use limits the size of the blocks to be transformed to a power of 2 points in the range 16 to 262144. You set the FFT block size from a drop down list. The result window ends up with half as many bins as the FFT block size. When you use the Process command, the selected area must hold at least as many points as the block size, otherwise no analysis is done.

The result window spans a frequency range from 0 to half the sampling rate of the waveform channel. The width of each bin is given by the waveform channel sampling rate divided by the FFT block size. Thus the resolution in frequency improves as you increase the block size. However, the resolution in time decreases as you increase the block size as the larger the block, the longer it lasts.

Windowing of data

- No Window
- Hanning
- Hamming
- Kaiser 30 dB
- Kaiser 40 dB
- Kaiser 50 dB
- Kaiser 60 dB
- Kaiser 70 dB
- Kaiser 80 dB
- Kaiser 90 dB

The mathematics behind the FFT assumes that the input waveform repeats cyclically. In most waveforms this is far from the case; if the block were spliced end to end there would be sharp discontinuities between the end of one block and the start of the next. Unless something is done to prevent it, these sharp discontinuities cause additional frequency components in the result.

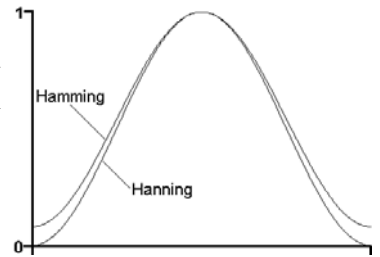
The standard solution to this problem is to taper the start and end of each data block to zero so they join smoothly. This is known as *windowing* and the mathematical function used to taper the data is the *window function*. The use of a window function causes smearing of the data, and also loss of power in the result.

You can find all sorts of windows discussed in the literature, each with its own advantages and disadvantages; windows shaped to have the smallest side-lobes spread the peak out the most. By reducing the side-lobes you decrease the certainty of where any frequency peak actually is (or the ability to separate two peaks that are

close together). Spike2 implements the following windows:

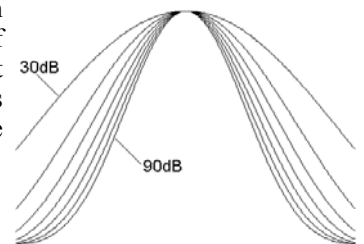
No Window You might use this if there is one sine wave, or if more than one, they all have similar amplitude. This has the sharpest spectral peaks, but the worst side-lobes.

Han or Hanning This is a good, general purpose, reasonable compromise window. However, it does throw away a lot of the signal. It is sometimes called a “raised cosine” window and is zero at the ends. If you are unsure about which window would be best for your application, try this one first.



Hamming This preserves more of the original signal than a Hanning window, but at the price of larger side-lobes.

Kaiser These are a family of windows calculated to have known maximum side-lobe amplitude relative to the peak. Of course, the smaller the side-lobe, the more signal is lost and the wider the peak. We provide a range of windows with side-lobes that are from 30 to 90 dB less than the peak.

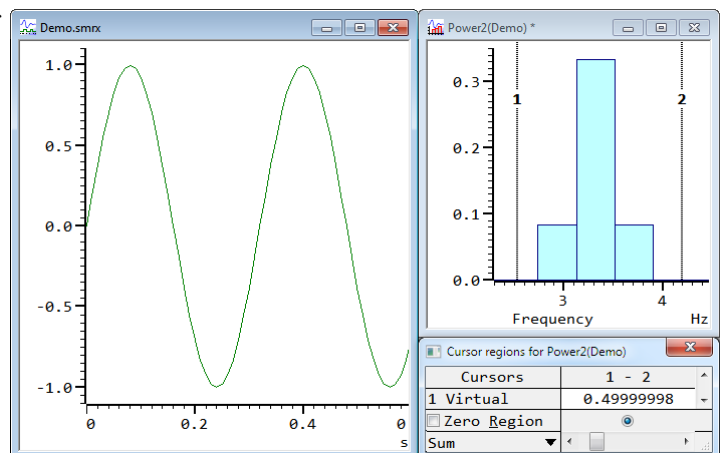


Overlapping data blocks

From Spike2 version 7.10, each source data block overlaps the previous, contiguous block by 50%. This is sometimes referred to as Welch's method. Previously, blocks were not overlapped, which is sometimes called Bartlett's method. Overlapping the blocks mitigates the effect of under-weighting data at the start and end of each block, especially with the Hanning and Hamming windows, where it weights all data the same. However, it also slows down the processing by up to a factor of two. You can disable the overlap in the Edit menu Preferences in the Compatibility tab. When overlapping blocks, we will increase the overlap, when needed, to make sure that all data in the range contributes to the power spectrum.

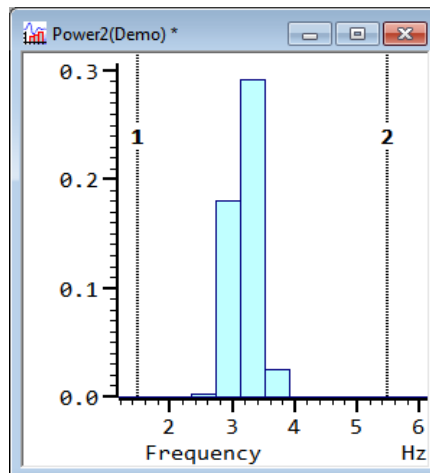
Power spectrum of a sine wave

If you sample a pure sine wave of amplitude 1 Volt and take the power spectrum, you will not get all the power in a single bin. You will find data spread over a few bins, and the sum of these bins will be 0.5 Volt². The factor of 2 in the power is because we give the result as RMS (root mean square) power. This is illustrated by the example below where we have created a 1 Volt sine wave using a virtual channel sampled at 100 Hz with the expression $WSin(3.125,0)$. We have formed the power spectrum of the signal using a 256 point transform using a Hanning window and zoomed in around the bins where the result lies.



We have used the cursor regions dialog to sum the power, getting 0.4999998, which is as close to 0.5 as one could hope for. If you repeat this and change the cursor regions measure to Mean in X, you will get a result of 3.125, which is the frequency we started with. We chose 3.125 Hz for this example so that the side lobes of the result were symmetrical. If you change the window from Hanning to No Window you will get an even better result with all the power in a single bin, this is because at 3.125 Hz with data sampled at 100 Hz, 256 points spans exactly 8 cycles of the data.

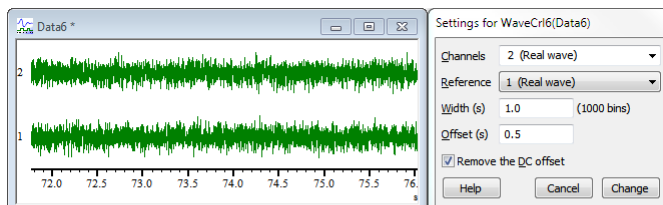
The image to the right shows the same measurements made with the frequency set to 3 Hz. In this case, the sum of the power is reported as 0.49999802 and the Mean in X value is 3.000024 Hz, still very much in agreement with the original data. In this case, if you set No Window, 256 data points contains 7.68 cycles, and the FFT will see this as a discontinuity in the data, resulting in spurious frequency components.



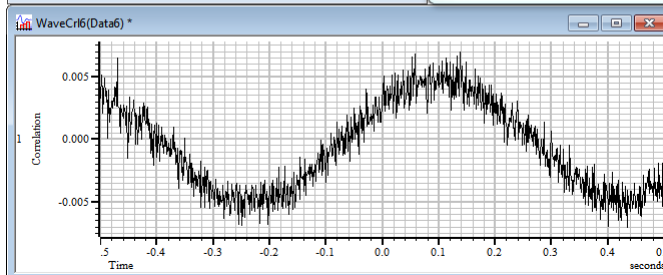
If you need access to the real and imaginary components of the FFT, see the `ArrFFT()` script function.

Waveform correlation

This command creates a result view that holds the correlation between two waveform channels sampled at the same rate or the auto-correlation of a channel with itself. The correlation measures the similarity of two waveforms. In the example below, the two waveforms hold random noise to which has been added a low amplitude 1.5 Hz sine wave. The sine wave in channel 2 is shifted forward by 0.1 seconds compared to channel 1, hence the peak in the correlation is at 0.1 seconds.



The correlation is calculated by multiplying the two waveforms together, point by point, and summing the products. The sum is normalised for waveform amplitudes and the number of points. This produces one result. The reference waveform moves one point to the right and the process is repeated to produce the next result. This is repeated for all the result bins. The results range between 1.0, meaning the waves are identical (except for amplitude) through 0 (uncorrelated) to -1.0, meaning identical but inverted.



The bins in the output are the same width as the sampling interval of the two input waveforms. You can choose to remove the DC component from the signals before calculating the correlation. This can be important as DC offsets can dominate the result. The correlation is calculated in such a way that you can change the setting of the DC offset removal without recalculating the correlation. To do it, open the **Process Settings...** dialog again, click in the box and then click the **Change** button.

The bins in the output are the same width as the sampling interval of the two input waveforms. You can choose to remove the DC component from the signals before calculating the correlation. This can be important as DC offsets can dominate the result. The correlation is calculated in such a way that you can change the setting of the DC offset removal without recalculating the correlation. To do it, open the **Process Settings...** dialog again, click in the box and then click the **Change** button.

The Process dialog sets the time range of the reference waveform. The analysis is only done for regions of data in which both waveforms exist. If you calculate correlations over long sections of data, the calculations will take some considerable time! For this reason we do not recommend this as an on-line analysis (although there is nothing to stop you using it). The number of calculations (and hence the time taken) is proportional to the number of points in the result times the number of points in the reference waveform.

Process settings

This command opens the analysis setup dialog of the current result or XY window. The window must have been created by the Analysis menu **New Result** view or **Measurements** command. It is the same as the dialog that created the window except the **New** button is now a **Change** button. The **Change** button is enabled if you change any field in the dialog and accepts changed settings and clears the result.

Process

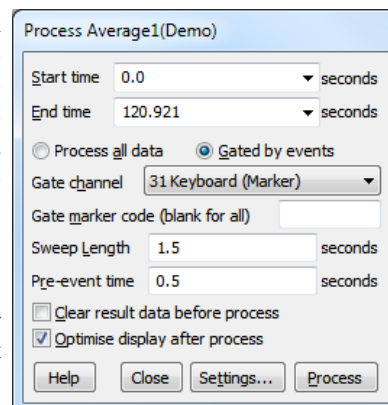
This command opens the Process dialog when the active window is a result or XY view attached to a time view. The dialog is also used to process measurements to a channel in a time view. It prompts you to select a time range of the data document to process. You can choose to add to the results of previous analyses, or you can clear the output. You can also choose to optimise the y axis of the result after the analysis is complete. The Settings button takes you back to the Settings dialog.

Start time and End time

These fields set the time range to process. For triggered analyses, for example Waveform averages, peri-stimulus histograms and event correlations, the time range sets the trigger points to use. For non-triggered analysis, for example power spectra and measurements, the time range sets the data to analyse.

If a triggered analysis has Manual as the trigger channel, then the Start time is used as the trigger for a single sweep. In the special case of a Phase histogram with a Manual trigger, both start and end times are used to mark the beginning and end of a single cycle.

If there is more than one time window associated with the data document, then the drop down lists include the window number. For example, View(2).Cursor(1), meaning the position of cursor 1 in the second duplicate time window.



Gated analysis

You will normally want to Process all data in the time range. However, you can also chose to process only those sections of the time window that lie within a user-defined time of particular events. For example, you might use events to mark a specific treatment type. You could then analyse data that fell within a time range of the treatment. To do this, select Gated by events and four extra fields are enabled.

Gate channel and Gate marker code

You must choose an event, marker or marker-derived channel to define the gate times. If the channel holds markers, you can specify a marker code as a two digit hexadecimal code or as a single printing character. Only events in the time range that match the code will be used as gates. Leave the field empty to use all markers in the time range in the current channel marker filter mask.

Sweep length and Pre-event time

These two fields define the length of time to process around each gate event and how far before each event to start processing. If gate events are closer than the sweep length, the sweeps are merged; the data is processed once only, not once per overlapped gate event. These fields are set in seconds, but you can provide the values in milliseconds or microseconds by appending ms or us to the value. For example, 0.5, 500ms and 500000us are all the same value.

Processing

When you click Process, Spike2 evaluates the start and end fields and processes the data selected by the dialog. The dialog remains on screen until removed with Close. This allows you to accumulate the results of processing different areas. When processing to a time view channel that is not a memory channel, you can append data at the end of the channel only; you cannot add new data before the end of the channel.

Check the Clear xxxx before process box to clear all result data before the results of processing are added. The xxxx depends on the type of processing. Check the Optimise Y axis after process box to rescale the results to display all values.

Breaking out of Process

Processing operations can take quite a time, especially in large data documents. If Spike2 detects a lengthy process operation, it displays a progress dialog in which you can cancel the operation. You can also stop processing early with the Esc key.

Process command with a new file

When the current window is a result or XY window attached to a sampling data file, the Process command activates a modified version of the process dialog. This dialog also appears when you create a result or XY window while sampling. It is also used to control processing measurements to a channel during sampling.

This dialog gives you control over when and how the result is updated. You can select Automatic, Gated by events or Manual updates. The dialog contents depend on the update mode. The two check boxes operate in the same way as in the Process dialog, described above. The Settings button takes you to the Process Settings dialog, Close removes the dialog. Apply and OK both apply the settings; OK closes the dialog.

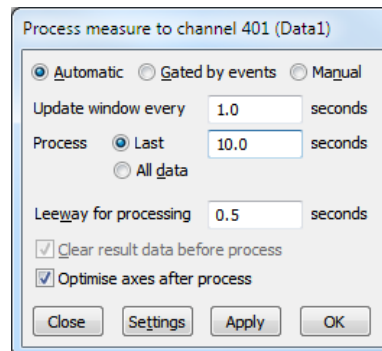
Automatic mode

In Automatic mode, the result updates as close to a user-defined interval as possible. Set the interval to 0 for the most frequent updates. Updates occur when the system has idle time.

The Clear before process check box is grey as each of the two Process modes has only one useful setting. The modes are:

Last Re-calculates the result for the time period set using the most recent data. Use this mode to follow changes. Clear before process is always checked.

All The result of processing new data is added to the analysis. Clear before process is always unchecked.

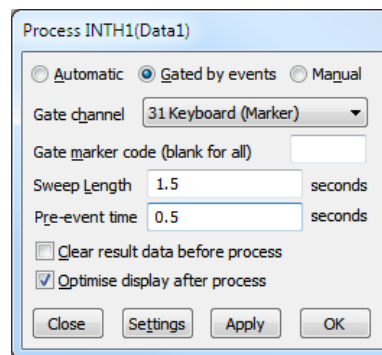


Leeway for processing is visible when processing with active cursors to an XY or time view channel. It sets the time to allow after cursor 0 for other cursors and measurements. If you set this to 2 seconds, the cursor 0 iteration is limited to 2 seconds back from the current time, allowing 2 seconds for other cursors and to take measurements. If your analysis does not generate any data, check that this value is large enough.

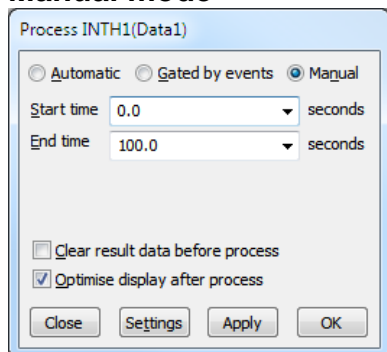
Gated by events mode

Gated by events mode analyses data around a specific gating event. You specify the sweep length, the start point of the analysis region relative to the gate event, the channel to use as a gate, and if this channel is a marker, the marker code to cause processing (see the Sampling data section for details of marker codes). To accept all marker codes set a blank Gate marker code. If you do not want the results to accumulate for all analysis periods, check the Clear before process box in the Process dialog.

Consider the case where you are generating a waveform average in response to a stimulus and you want to see how the average differs in the presence of inhibitors and a control. You could set up separate stimulus channels for each inhibitor and the control, but it is often easier to leave the stimulator running continuously and use the event gating to mark the regions for a particular treatment.



Manual mode



If you select Manual updates you must provide a start and end time for analysis. If the End time you set has already been reached, the data will be processed as soon as you click Apply (process and leave the dialog open) or OK (process and close the dialog).

However, if the End time is yet to occur, there is no processing on Apply or OK until the End time is available in the data document. At that point, the data will be processed and the target window will update. If you wish to process a different area, set a new time range and click the mouse on the Apply or OK button again.

Manual mode is the most similar to working with with the Process dialog offline. However, during data capture you are usually quite busy and it is often easier if you choose one of the other, more automated ways to select data for analysis.

Measurements to XY views

You can generate trend plots in an XY view both on-line in real time and off-line. Use the **Measurements** menu item and select **XY view**. This command generates a new XY window with one or more channels of data derived from the current time view.

The basic idea is that cursor 0 steps through the data following a user-defined rule. This can be as simple as *move to the last cursor 0 position plus 1 second* or it can be a complex data-searching algorithm. Each cursor 0 move also triggers the other active cursors to search. For each cursor 0 position, we take an x and a y measurement and pass the (x,y) point to a channel in the XY view. Using this mechanism, a wide variety of measurements can be made and displayed.

A **Process** command sets the data region to search for cursor 0 step points, exactly as for the result view analysis commands. You can process data both on-line and off-line. Processing can take a while if there are many cursor 0 steps to do. A progress dialog appears in a long process to allow you to stop early.

Cursor 0 stepping

This dialog area controls how cursor 0 moves through the data. If cursor 0 is already set to a suitable active cursor mode, this mode is copied to the dialog. You can set any active cursor mode that can iterate through data. These are: **Peak find**, **Trough find**, **Rising threshold**, **Falling threshold**, **Outside thresholds**, **Within thresholds**, **Slope peak**, **Slope trough**, **Turning point**, **Data points** and **Expression**. The other fields in this section depend on the stepping mode.

You can use the **Ignore cursor 0 step if** field to reject a cursor 0 step and step again. If this field is not blank and evaluates without error as true (not zero), the current cursor step is skipped. Expressions usually involve cursor values, for example $\text{Cursor}(0) > \text{Cursor}(1)$. You will normally leave this field empty.

If you check the **User check positions** box, you are given the opportunity to adjust the cursor positions after each step and before each measurement. This is ignored if the data file is being sampled.

Plot Channel

The trend plot can generate up to 32 channels of data in the XY view. The dialog starts with one channel. You can set the channel title in the **Plot Channel** box and create additional channels with the **Add Channel** button. The **Delete Channel** button deletes the current channel; you cannot delete the last channel.

X and Y measurements

These two areas have identical functionality. They generate the x and y values that are passed to the XY view for each point. The fields displayed in these areas depend on the measurement **Type** field. Most measurement types depend on times. For a useful result, you will set these to times relative to cursor 0, or to an active cursor that was positioned relative to cursor 0.

All channels use same X

With multiple plot channels check this box to use the same x measurement for all channels. This is most commonly used when the x measurement is a time.

Points

Set this field non-zero to limit the displayed XY points. Points are added up to this limit, then each new point causes the oldest to be deleted. This is most useful on-line, for example to show pressure-volume curves or to display eye movements when you have horizontal and vertical input data.

New and Cancel

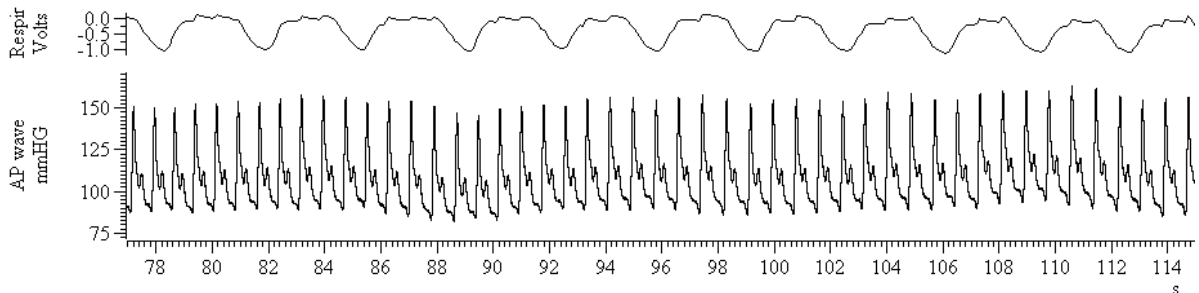
The New button creates the XY view and opens the Process dialog, ready to set a time range to step cursor 0 through to process the data. If you return to this dialog after creating the XY view, New becomes Change. Cancel closes the dialog.

Tabulated output

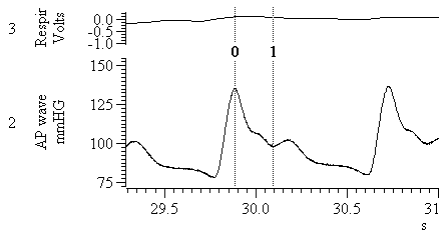
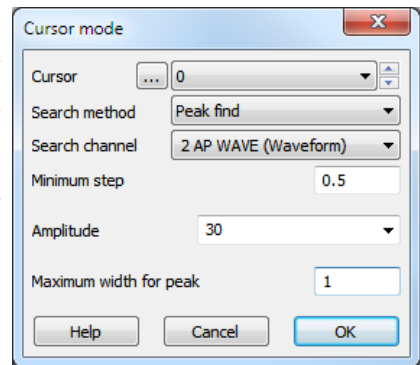
You can tabulate the results in an XY view with the Edit menu Copy as Text command.

Example plot

As an example of a trend plot, consider this data file containing an arterial blood pressure on channel 2 and a respiration signal on channel 3. Let us suppose that you are interested in the position of the dichotic notch (the small downward blip after the peak of the blood pressure) relative to the peak of the blood pressure.

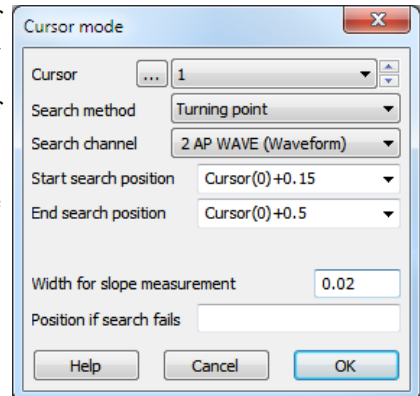


The first step is to decide how to step through the data. The obvious method is to locate the blood pressure peaks. I used the Cursor menu Active modes... command to select cursor 0, set the search method to Peak find, the search channel to 2 and Amplitude to 30. Any Amplitude from 5 to 40 would work as each cycle is at least 40 mmHg from peak to trough and the biggest wobble in between is around 5 mmHg. I set the Minimum step to 0.5 and the Maximum width for peak to 1 to reject artefacts (although this data does not have any). I checked that this was working with Ctrl+Shift+right and cursor 0 stepped from peak to peak.



The next step is to locate the notch. I added cursor 1 to the window with Ctrl+1, and then opened the cursor Active modes dialog for cursor 1. This time I set the method to Turning point, set channel 2, set the search range to start at Cursor(0)+0.15 and end at Cursor(0)+0.5 and set the Width for slope measurement to 0.02 seconds.

I used Turning point mode and not Trough because the amplitude of the peak after the notch may be very small. I started the search just after the peak because the peak is also a turning point (where the slope changes sign) and I wanted to exclude it. From a visual inspection of the data I could see that the notch was always more that 150 ms past the peak, so by starting the search at Cursor(0)+0.15 I avoided the possibility of detecting the change of slope between the peak and the notch.



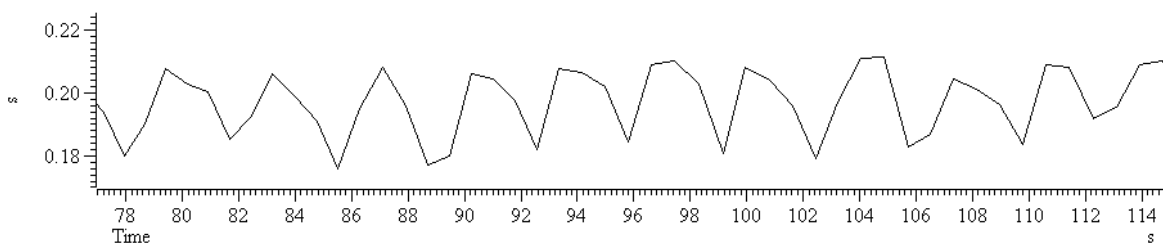
The end of the search is set so that any reasonable notch will be found. The hardest item to set is the Width for slope measurement. This needs to be big enough so that noise in the signal doesn't cause false turning points, but small enough so we don't miss a small one. I checked it was working with Ctrl+Shift+right and now both cursors stepped along the data. Position if search fails is left blank so that if the turning point cannot be found, no measurement is taken.

The final step is to generate the graph. I used the Analysis menu Measurements command and selected XY View. When the dialog opens it automatically picks up any active cursor setting from cursor 0.

I wanted to plot the notch position relative to the peak against the position of the peak. To do this I set the X measurement to be Time at point and selected Cursor(0) to be the point. I set the Y measurement to Time difference and set the Reference time to Cursor(0) and Time to Cursor(1) to form $\text{Cursor}(1) - \text{Cursor}(0)$ as my measurement.

All that remained was to click New to create the XY window and open a Process dialog. I adjusted the time window so that it displayed the data I wanted to process, then

set the start and end times to $x_{\text{Low}}()$ and $x_{\text{High}}()$, selected Process all data and clicked the Process button. The picture below shows the result. For each heart beat in the time range, we have a plot of the time delay from the peak to the dichotic notch.



Of course, we could have made a wide variety of measurements. The x axis need not be time, it could be a value derived from a different channel. For example, looking at the results, we can see that there seems to be a link between the notch position and the respiration signal, so we might have set the x axis to the value of the respiration channel at the cursor 0 time.

Measurement fields

The X measurements and Y measurements areas determine the x and y values passed as a pair into each XY channel. The Type field sets the measurement method. The contents of the measurement area depend on the measurement type. The following fields are used:

Channel

The data channel used for a measurement. Select the data channel from the list.

Time

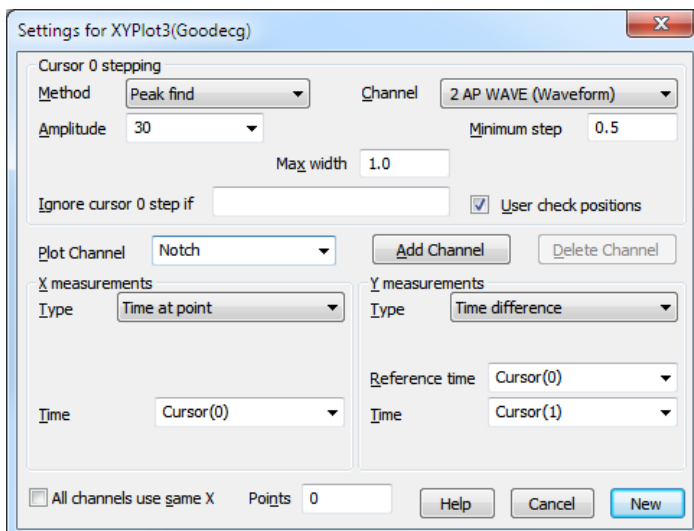
This is a time for use either as a value, or the time at which a measurement of the data channel is to be made. This field will usually be set to an expression that contains a cursor position, for example $\text{Cursor}(0) - 1.3$ or $(\text{Cursor}(0) + \text{Cursor}(1)) / 2$. You can select expressions from the drop down list.

Expression

This is used for Expression measurement mode only and is the expression to be used for a measurement. This will usually be a view expression that evaluates to a time.

Prompt

This is used in User entered value measurement mode only, and is the prompt to use to request data values from the user.



Reference time

This field is used in the same way as the **Time** field. It specifies a time to subtract from the time in the **Time** field for the **Time difference** measurement, or the time at which to measure the data value to subtract for the **Value difference** measurement.

Width

This field is used to set the width of a point measurement around a time. If you set this to 0, the nearest data point is used, otherwise a mean value over the time range from $\text{Time}-\text{Width}/2$ to $\text{Time}+\text{Width}/2$ is used.

Start time, End time

These two fields always occur as a pair and identify a time range in which to take a measurement. Both fields will accept an expression for a time and you can select likely expressions from the drop down lists.

Minimum step

Cursor 0 does not use the **Start time** and **End time** fields. Instead, you set the minimum step from the last cursor 0 position. The search range for cursor 0 ends at the end of the file (or the start if you are searching backwards) and starts at the minimum step away from the last cursor 0 position.

Measurement types

In the description of each measurement type we refer to waveforms and events. By waveform, we mean either a waveform channel, or a WaveMark channel drawn as a waveform. The measurements are:

Value at point

This is the value of the nominated **Channel** at the specified **Time**. If **Width** is non-zero, the measurement is the mean value over the time range from $\text{Time}-\text{Width}/2$ to $\text{Time}+\text{Width}/2$. For a waveform or a channel drawn as a rate, mean frequency or instantaneous frequency, the result is in y axis units. For all other event drawing modes, the result is the time of the first event at or after **Time**.

Value difference

This is the value of the nominated **Channel** at **Time** minus the value at **Reference time**. If **Width** is non-zero, the measurement is the mean value from $-\text{Width}/2$ to $\text{Width}/2$ around each time point. For a waveform or a channel drawn as a rate, mean frequency or instantaneous frequency, the result is in y axis units. For all other event drawing modes, the result is the time difference between the first event at or after **Time** and the **Reference time**.

Value above baseline

This is the same as **Value difference**, except that the **Width** measurement is applied only to the reference time value. The expectation is that the reference time is set in the middle of a baseline region, and value is a spot value at **Time**.

Value ratio

This is identical to the **Value difference** measurement except that instead of subtracting the values, we divide the value of the nominated **Channel** at **Time** by the value at the **Reference time**. Attempted division by zero does not produce a measurement.

Value product

This is identical to the **Value difference** measurement except that instead of subtracting the values, we multiply the value of the nominated **Channel** at **Time** by the value at the **Reference time**.

Time at point

This is the value of the expression in the **Time** field. This is often a cursor position, for example: `Cursor(0)`.

Time difference

This is the value of the expression in the **Time** field minus the value of the expression in the **Reference time** field. In many cases, both these values will be cursor positions.

Fit coefficients

A fit is requested on the channel set by the **Channel** field after each cursor iteration and the value of the coefficient selected by the **Coefficient** field is the measurement. For this to be useful, you must define a fit for the channel using iterating cursors for the start and end times. If a valid fit exists for the channel you can choose the coefficient by name, for example: **a0**: Amplitude, **a1**: Time constant, **a2**: Offset, otherwise you select the coefficient as **a0**, **a1**, **a2** and so on.

Expression

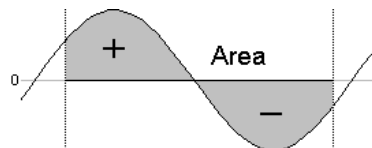
This is an expression (usually involving cursor positions) that is evaluated at each step to produce the value.

Use entered value

You will be prompted to enter a value at each step of the cursor 0 position. If you enter a non-blank prompt, this is used when you are asked for a data value.

Area

If the **Channel** is drawn as a waveform (has a y axis), the result of this measurement is the area between the waveform channel and the y axis zero over the time range set by **Start time** and **End time**. The area is positive for curve sections above zero and negative for sections below zero. For all other cases this is the count of events in the time range.



Mean

For channels drawn as waveforms, the result is the mean value of the data points between **Start time** and **End time**. For all other channel types the value is the number of events in the time range divided by the time range. This could be thought of as the mean event rate in the time range.

Slope

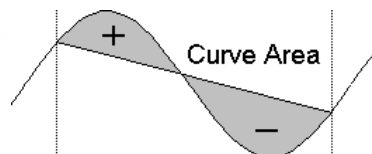
This has meaning only for a channel with a y axis. The result is the slope of the best-fit straight line by the least squares method to the data between **Start time** and **End time**.

Sum

For a channel drawn as a waveform (with a y axis) this is the sum of the values of the data points. For all other cases it is the same as the Area measurement.

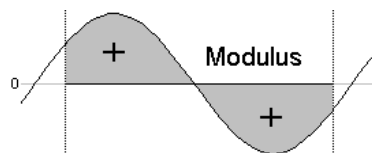
Curve area

This measurement only has meaning for channels that are displayed with a y axis, such as waveforms or mean frequencies. It is the area between the straight line joining the ends of the waveform data in the time range and the data points. Curve areas above the line add to the result; areas below the line subtract from the result.



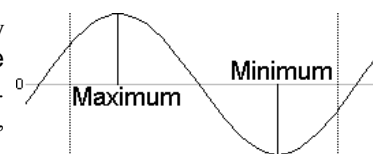
Modulus

This has meaning only for channels drawn as waveforms (with a y axis). It is the same as the Area measurement except that all areas (above and below the zero line) count as positive. For channels drawn as events there is no measurement.



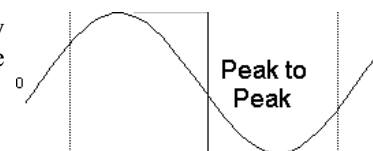
Maximum, Minimum, Extreme

These have meaning only for a channel drawn as a waveform. They measure the maximum and minimum values in the time range and **Extreme** measures the larger of the maximum and minimum value ignoring the sign. For example if the maximum value was 6 and the minimum value was -7, the extreme value would be 7.



Peak to Peak

This measurement has a value for channels drawn as a waveform (with a y axis). It is the difference between the maximum and minimum value of the channel in the time range **Start time** to **End time**.



RMS amp

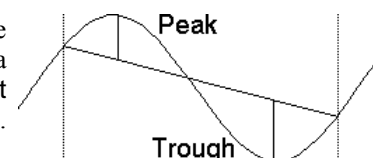
This measurement has a value for a channel drawn as a waveform. It is calculated by summing the square of each data point, dividing the sum by the number of data points and then taking the square root of the result.

SD

The standard deviation has a value only for a channel drawn as a waveform. It is calculated by finding the mean of the data, then summing the squares of the differences between each data point and the mean, dividing the sum by the number of data points minus 1, and finally taking the square root of the result.

Peak and Trough

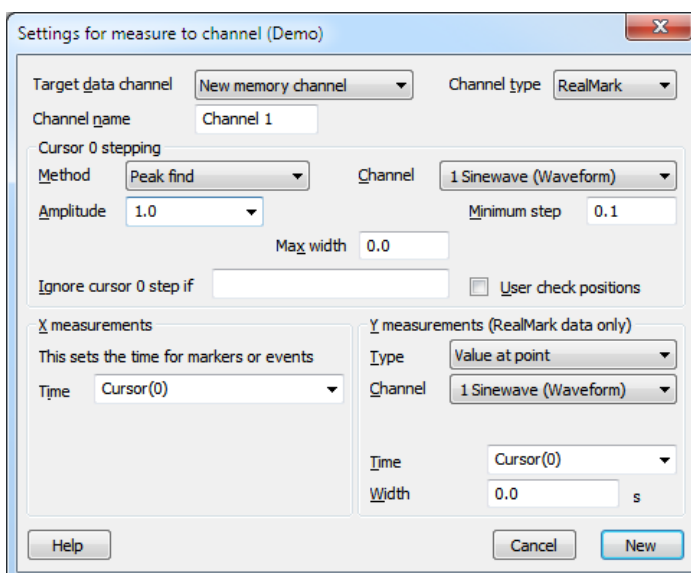
These measurements only have meaning for waveforms. These values are the maximum positive and negative distances between the waveform and a straight line joining the end points of the waveform in the time range **Start time** to **End time**. The **Peak** value is always greater than or equal to 0. The **Trough** value is always less than or equal to zero.



Measurements to a data channel

This command generates a new event, marker or RealMark data channel in the current time view. The new channel holds the results of measurements made with the active cursors. You can use this command on-line as well as off-line. For example, if you were recording a blood pressure or ECG signal, you could use this to pick signal peaks to provide a real-time heart rate channel, directly from the blood pressure signal.

Use the **Measurements** menu item and select **Data channel** to open the dialog. You can also activate this dialog by right clicking on a channel that was created by a measurement and selecting **Process Settings** from the channel pop-up menu.



Target data channel

When you are creating a new channel this field holds a drop-down list of possible channels. You can choose an unused disk channels or **New memory channel**. If you open the dialog after creating a channel, the field holds the channel number. You cannot replace an existing channel from this dialog; you must delete the existing channel first.

Channel type

Choose from RealMark, Marker or Event data. If you change the channel type after processing this will delete all the channel data when you click **Change**. For RealMark data, the target channel has one real value attached to each marker. The value is set by the Y measurements field.

Channel name

This field sets the name of the new channel. For RealMark data, the channel units are copied from the Channel used for the Y measurement.

Cursor 0 stepping

The fields in this dialog area are identical to those in the Measurements to XY views.

X measurements

The Time field in this dialog section sets the time of each item that is added into the data channel. You will normally type in an expression that is related to cursor positions.

Y measurements

The fields in this dialog area are identical to those in the Measurements to XY views. You can use this section of the dialog when the type of the target channel is RealMark otherwise this section is disabled. The value extracted here sets the RealMark data value.

New/Change

This button is labelled **New** when you are creating a channel and **Change** if you return to this dialog after creating the channel. If you change the channel type and click **Change**, any previous data stored in the channel is deleted. You are asked to confirm that this is what you intended to do.

When you click the **New** or **Change** button, the **Process** dialog opens for you to choose the time range to process to generate data for the new channel. If you are sampling data into the time view, the on-line version of the **Process** dialog opens.

Fit Data

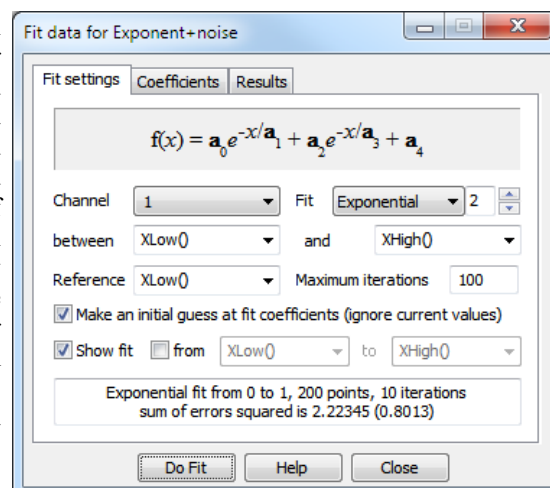
This command opens a tabbed dialog from which you can fit mathematical functions to channels in a time, result or XY view. In a time view you can fit to an event-based channel as long as the data uses a display mode that has a y axis. If you fit data to a channel in a result view and error bars are displayed, the fit minimises the chi-squared value, otherwise the fit minimises the sum of squares of the errors between the data and the fitted curve. In addition to best-fit coefficients and an estimate of how much confidence to place in them, you also get an estimate of how likely it is that the model you have fitted to your data can explain the size of the chi-squared value or sum of errors squared. If your data does not have error bars, these estimates are based on the assumption that all data points have the same, normally distributed error statistics. The dialog has three tabs:

Fit settings Set the fit type and range of data to fit and range to display

Coefficients Set the starting point for your fit and optionally fix coefficients

Results Display the fitting results and residual errors

The three buttons at the bottom of the dialog are common to all pages. The **Help** and **Close** buttons do what they say. **Do Fit** attempts to fit with the current fit settings.



Fit settings

This page of the Fit Data dialog controls the type of fit, the data to fit and what to display. The area at the bottom of the window gives a synopsis of the current fit state. Fields are:

Channel

You can select a single channel from the current view. If this is a time view, the channel must have a y axis. If you change the display mode of an event-based channel, any fit associated with the channel will most likely become invalid. In a result view, the fit is done to the basic channel data, even if the channel shows raster data.

Fit

The fit to use is defined by its name and the order of the fit (a number). For example, an exponential fit allows single exponents or double exponents. The window at the top of the dialog displays the mathematical formula for the fitting function. The following fits are currently supported (N is the maximum order allowed):

Name	N	Comments
Exponential	2	This fit includes an offset. You can force a zero offset in the coefficients page.
Polynomial	5	These fits do not require starting values for the coefficients.
Gaussian	2	If you attempt to fit two overlapping peaks you may need to manually adjust the guesses for the peak centres to get convergence.
Sine	1	You can fit a single sinusoid with an offset. If the frequency guess (in radians) is not reasonably close, the fit may not converge.
Sigmoid	1	This fits a single Boltzmann sigmoid by an iterative method.

Range

You fit data over a defined x axis range, set by the **between** and **and** fields. You can choose values from the drop down list or type in simple expressions, for example `Cursor(1)+1`. There must be at least as many data points to fit as there are coefficients. For example, to fit a double exponential, which has 5 coefficients, you need at least 5 points. Most fits will use many more points than coefficients.

Reference

This is the x axis position to use as the zero value of x in the fitting function. The most common value for this would be the start point of the fit. However, in some cases you may want this to be elsewhere. For example, in exponential fitting, you may want to calculate the amplitude of a trace at some position. Making this position the reference point makes it easy to calculate the amplitude (it is the sum of the even-numbered coefficients).

Maximum iterations

All fits except the polynomial are done by an iterative process. Each iteration attempts to improve the coefficient values. The iterating stops when improvements in the fit become insignificant, the iteration count is exceeded, the mathematics of the fitting process suggests that the fit is not going to improve or there is a mathematical problem. This field sets the maximum number of iterations to try before giving up.

Make an initial guess

The iterative fits need a starting point. There are built-in guessing functions that usually generate a starting point near enough to the solution that the fitting process can converge. If you check this box, these guessing functions are used each time you click the **Do Fit** button. Otherwise, each fit starts with the current values.

Show fit

Check this box to display the current fit. If the **From** box is checked, you can also choose the range over which to display the fitted data. If this box is not checked, the fit is displayed over the fitted data range.

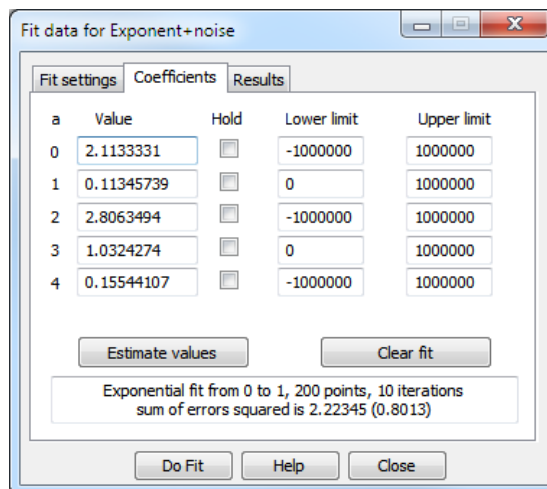
Coefficients

This page of the Fit Data dialog lets you set the starting values for iterative fits. You can also use this page to hold some of the coefficients to fixed values and you can set the allowed range of values for fitting.

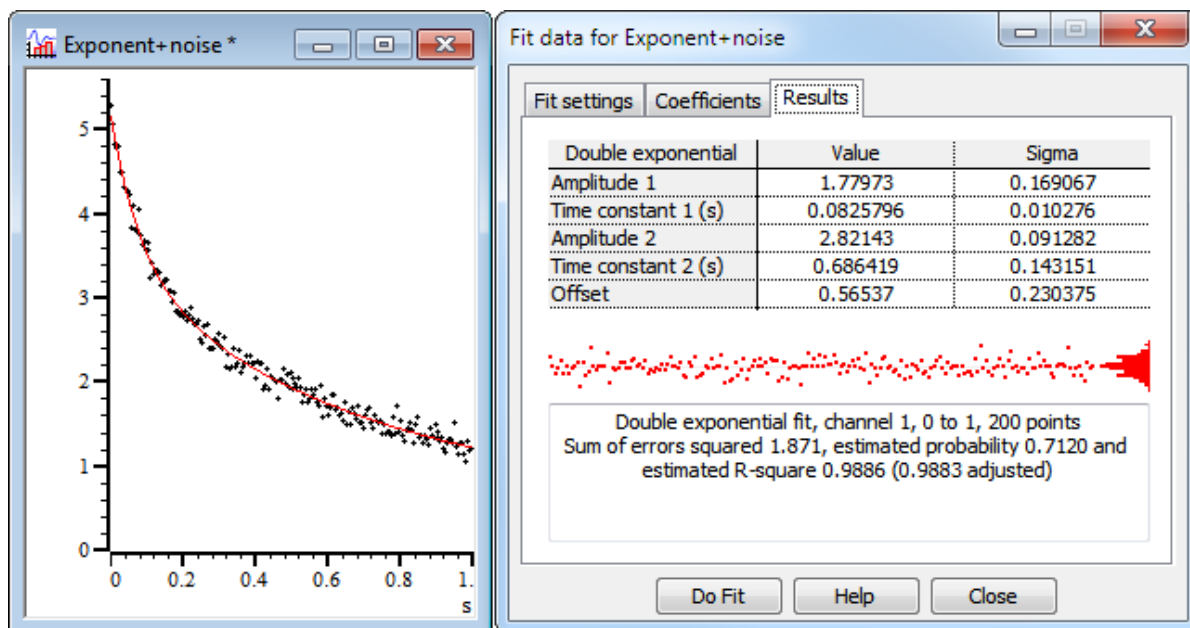
If you know the value of one or more of the coefficients, type the value in and check the HOLD box next to it. For example, in an exponential fit you may know that the final coefficient (the offset) is zero.

The limit values are applied after each iteration. The fit may have to follow a convoluted path before it converges on a solution, so do not set the fit limits too close to an expected solution as this may prevent convergence.

The Estimate values button can be used to guess initial values for fitting based on the raw data. The Clear fit button removes the fit from the channel.



Results



$$Fitting\ to\ y = 2.0 * exp(-x/0.1) + 3 * exp(-x/1.0) + RandNorm(.1, 0)$$

The results page of the Fit Data dialog holds information about the last successful fit. The page has three regions: coefficient values at the top, a message area at the bottom, and a plot of the residuals (differences between the fit and the data) in the middle. The residuals are displayed immediately after a fit but will not be displayed if you close the dialog and reopen it.

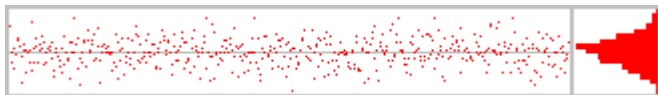
Coefficient values

The Value column holds the fitted value that minimised the chi-squared or sum of squares error for the fit. The Sigma column is an estimate of how the errors between the fitted curve and the original data translates into uncertainty in the fit coefficients given that the model fits the data and that the errors in the original data are normally distributed. If a coefficient is held, the Sigma value will be 0. The Testing the fit section gives more information on the derivation of these values and how to interpret them. You can select rows, columns or individual cells in this area, the use Ctrl+C to copy them to the clipboard. This also copies a bitmap image of the page to the clipboard.

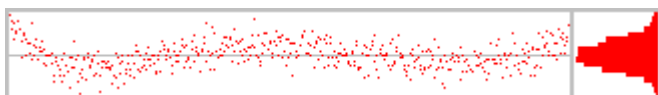
Residuals

This section of the page displays the differences between the data points and the fitted curve in the large rectangle and a histogram of the error distribution on the right. The error plot is self-scaling based on the distribution of errors; the plot extends from +3 at the top to -3 at the bottom times the RMS (root mean square) error. The grey line across the middle of the plot indicates an error of zero.

In the case that the data can be modelled by the fitting function plus normally-distributed noise, you would expect to see residuals distributed randomly around the 0 error line and the histogram on the right should resemble a normal curve.



If the data cannot be modelled in this way, you would expect to see evidence of this in the residuals. In this example (generated by fitting a cubic to data that was actually a double exponential), you can see that there are clear trends in the errors.



In extreme cases, the error due to the wrong model being used becomes much larger than the errors due to uncertainty in the data values, and you get a residual plot like this one.



Message area

This area displays a summary of the fit information that you can select with the mouse and copy to the clipboard. The first line holds the type of the fit, the channel number, the ordinate range and the number of points in this range. For example: "Double exponential fit, channel 1, 0 to 50, 50 points".

If you are fitting a result view channel that has error information displayed, the second line displays the chi-squared error value for the fit and the probability that you would get a chi-squared value of at least that size if the function fits the data and the errors are normally distributed. For example: "Chi-square value 58.6, probability 0.5867, R-square 0.9886 (0.9883 adjusted)".

Otherwise, the second line holds the sum of the squares of the errors between the data and the fit and an estimate of the probability that you would get a sum of squares of errors of at least this size based on the assumptions that the errors in the original data had a normal distribution that was the same for all points. For example: "Sum of errors squared 1.871, estimated probability 0.7120, estimated R-square 0.9886 (0.9883 adjusted)".

If the probability value is very low or very high, there are extra lines of information warning that the fitted function plus normally-distributed noise is unlikely to model the data, or that the errors in the original data have probably been over-estimated. See Testing the fit for more information.

The R-square value is the standard "goodness of fit" value, which is the proportion of the variance of the data that can be explained by the fit, so a larger value is better. If you want to compare fitting different orders of polynomials or exponents you should use the adjusted values as these discount the improvement you would expect just because you increase the number of coefficient.

Context menu

If you right click on this page you are offered a context menu that contains Copy, Log and Log Titles commands. The Copy command copies selected sections of the results, or all the results if there is no selection to the clipboard as text. It also copies the page as a bitmap. The Log command prints a one-line synopsis of the current fit to the log window. The Log Titles command copies a suitable set of titles for the logged data.

Testing the fit

When you fit a model to measured data to obtain the best-fit coefficients, there are two questions you would like answered:

1. How well does this model fit the data? Put another way, how likely is it that this model plus some degree of random variation can explain my data set?

2. Given that the model does fit the data, how much confidence can I place in each of the fitted coefficient values?

When we talk about fitting curves to data, we are making the implicit assumption that you took measurements from some process that follows a model, and that this model can be expressed as a mathematical function with adjustable parameters, which are our fitting coefficients. Further, we assume that the measurements you make are not perfect; they have random variations with a known probability distribution about the correct value. To allow us to calculate likelihoods, we assume that this probability distribution is a normal (Gaussian) distribution. In the real world, of course, only some of this may apply. You may have no a priori knowledge of the distribution of errors in your original data, and this distribution may be anything but normal.

(Estimated) probability

This value is an attempt to say how likely the model you have chosen plus normally distributed noise is to explain the measured data. If the residuals of your data show other than a random distribution around the zero line, the probability will be 0 (or close to it).

R-square (R^2 or $R2$ or $r2$) value

This is a simple measure of the "goodness of fit" and describes what proportion of the variance of the original data is explained by the model. One would expect values in the range 0 (meaning that the model accounts for none of the variation) to 1 (the model accounts for everything). With non-linear fits, it is possible to get a negative result, meaning that a horizontal line through the mean of the data is a better fit than the model.

We also give an adjusted R-square value. This is useful when you have a model that allows you to choose the order of the fit and you are not sure what order is appropriate. For example, when fitting polynomials you have a choice of order 1 through 5. When you increase the order, you make it easier for the model to fit random changes in the data, so the R-square value will improve as the order increases. The adjusted R-square value is less than the R-square value by an amount that quantifies how much we would expect the R-square value to increase by chance due to having more fitting coefficients.

You can read more about R-square [here](#).

Chi-square fits

In the ideal case, where you know the standard deviations of each data point, the fitting minimises the chi-squared value, which is the sum of the squares of the differences between the model and the data points divided by the standard deviation (expected error) of the original data point values. Given a chi-squared value and the number of points it was measured from, we can calculate that probability of getting a chi-squared value at least this large, due to random variations in the data. This is the value given in the **Results** tab. Ideally, you would like to see a value around 0.5, meaning that you were equally likely to get a larger value as a smaller one. Values very close to 1 mean that, given the errors in each point, the data is too close to the model. Either the error estimates are too large, or the data has been "improved". Although you can hope for probabilities in the range 0.1 to 0.9, values down to 0.01 may occur for acceptable fits, and even smaller values can occur if your error distribution is not as normal as you thought.

Very low fit probabilities will occur if your data contains variations that are significant compared to the errors in the input values and that are not included in the model. For example, if you are fitting exponents to a sampled waveform that includes perceptible mains interference, you can get a good fit (by eye) to the exponential data, but with a probability of 0.0000 as far as the mathematics is concerned because the model does not include the mains hum and cannot explain why the chi-squared value is so high.



If we assume that the model fits the data, we can estimate the standard deviation of the fitted coefficients. This is the likely variation in the fitted coefficients if we re-ran the experiment many times and fitted the data to each set of results. This is presented as the **Sigma** value in the results tab.

Least-square fits

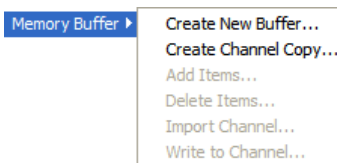
If there is no error information for each point, we assume that all the points have the same, normal error distribution and the fit minimises the sum of squares of errors between the model and the data. Because there is no independent estimate of the likely spread of the errors in the original data, strictly speaking, there is no way to give a probability of getting an error of at least this size.

However, we can say (though statisticians may shudder), "Given that the model does fit the data, and that the errors all have the same, normal distribution, then the differences between consecutive errors should also be normally distributed with twice the variance of the errors". We use this to estimate the standard deviation of the data and then we apply the probability test. We label this as estimated probability. The same comments about likely values apply as for the Chi-square fits, except that very small values may just mean that our estimation process fails for your data.

The coefficient Sigma values are calculated on the assumption that the model fits the data, that all the original points have the same standard deviation, and that the standard deviation of the original data can be deduced from the residual sum of squares errors.

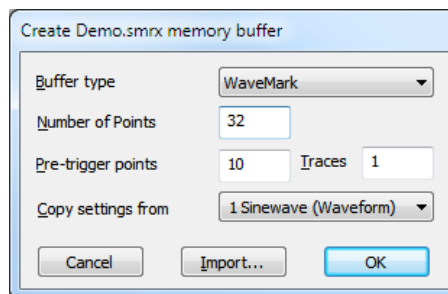
Memory buffer

Each data file has up to 300 memory channels. They hold data copied from existing channels, derived from waveform channels or entered by hand. You can display the memory buffers and use them like any other channels. The memory buffers can also be written to the data document. If you do not write them, the memory channels are lost when you close the file. Each memory buffer expands as events are added. The size is limited by available memory.



Create New Buffer

This command creates a new memory buffer channel of any type. The channel numbers are in the range 401 to 700. Previous versions of Spike2 used channels 101 to 200 and the numbers may change again in future versions. The memory channel numbers are displayed as m1 to m300. A new channel gets the lowest available number. You can use the Analysis menu Delete Channel command to remove memory buffers. OK creates the channel, Import creates it and opens the Import channel dialog.



Channels created with this command are not permanent. The data is kept in memory and is lost when the file is closed. If you need to make the data permanent, you must write it to a permanent channel with the Write to Channel option.

The fields in the dialog change depending on the type of buffer you choose to create. The example shown is for a WaveMark channel. The fields are:

Buffer type

You can create a channel of any type: Event-, Event+, Level, Waveform, RealWave, Marker, TextMark, RealMark or WaveMark.

Number of ...

This field is present for TextMark data where it sets the maximum number of characters to store with each mark, RealMark data where its sets the number of real values to store with each mark and for WaveMark data where it sets the number of waveform points to store with each mark.

Initial level

This field is present for Level event data and it sets the initial level of the channel before any events are added at Low or High. To invert an existing level event channel, create a memory channel with the desired initial state, then import events into it.

Pre-trigger points and traces

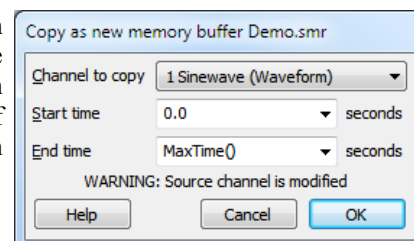
These fields are present for WaveMark data only and set the number of waveform points before the trigger and the number of traces in each WaveMark item in the range 1 to 4.

Copy settings from

When you create a waveform, RealWave or WaveMark buffer, Spike2 needs to know the waveform sample interval and calibration. This field indicates a channel that holds waveform, RealWave or WaveMark data and the buffer is given the same sampling rate and calibration. If you create a buffer from the script language you can choose a sample rate and calibration without reference to an existing channel.

Create Channel Copy

This command creates a new memory channel that is a copy of an existing channel. The source channel can be of any type. If the source channel has a channel process attached, or if it is a marker channel with an active marker filter, the new channel will contain modified data. If this is the case, a message appears in the dialog warning you. You can also activate this command by right-clicking on a channel.



Channel to copy

The source channel to copy. Copied items include the channel title, units and comment.

Start time, End time

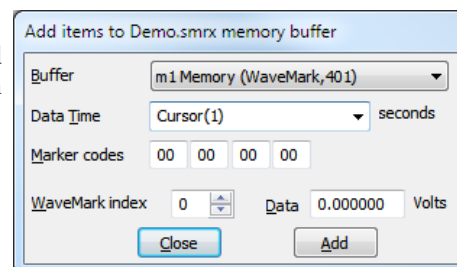
These fields let you choose a time range of the original data to copy to the new channel.

Add items to memory buffer

This command adds a data item to a memory buffer of any type. Click Add to place or replace data in the buffer at the specified time. Waveform and RealWave data is aligned in time with existing data. The fields depend on the channel type:

Buffer

The memory buffer channel to use.



Event time

The time at which to add data. If you have cursors enabled in the time view you can add data at the time of the cursor (as in the example). This field is present for all channels.

Marker

These four fields are present for all Marker derived data types. You can set the marker codes appropriate for your data here.

WaveMark/RealMark data

This field is present for WaveMark and RealMark channels. Enter the index of the data point that you wish to set in the Data field. There is also a Trace field for WaveMark buffers with multiple traces.

Wave value

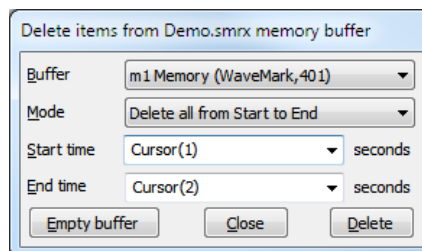
This field is present for waveform and RealWave channels. You can type in a number or an expression that can include horizontal cursor values (for example `HCursor(2)` or `H2`).

TextMark string

This field is present for TextMark data only. Enter the text to appear in the new data item.

Delete items from memory buffer

This command opens a dialog in which you can delete one or more data items or a time range from a memory buffer. The **Delete** button removes one or more items as set in the dialog. The **Empty buffer** button deletes all data for this channel (it does not delete the channel). **Close** removes the dialog. The `MemDeleteTime()` script command has the same functions.



Buffer

The memory buffer channel to use.

Mode

There are four modes: **Delete nearest to Start within Range**, **Delete all round Start within range**, **Delete first in range Start to End**, **Delete all from Start to End**. The first two modes delete one or more data items around a time, the other two modes delete the first or all data items in a time range.

Start time

The time range is **Start time – Time range to Start time + Time range** in the first two modes, and **Start time to End time** for the last two modes.

End time

The end of the time range. This field appears in the second two modes.

Time range

The time range around the **Start time** to search for data. This field appears in the first two modes. This field is usually set to a small value so that you can delete events close to the position of a cursor.

Import channel

You can import data into a memory buffer selected by the **Buffer** field from a channel set by the **Channel** field. You may not import data from a memory buffer to the same memory buffer. The **Start time** and **End time** fields set the time region to import data from.

The **Mode** field is present if the source is a waveform channel and the destination is not, and sets the method to extract event times from the waveform. The **Minimum interval** field and one of the **Size/Level** fields are present if **Mode** is present.

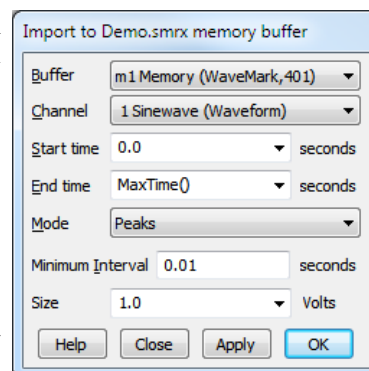
To import a waveform channel to a waveform or WaveMark channel the sampling rates must match. Apart from this restriction, you can move data from any channel to any buffer. During data import, values that cannot be extracted from the source are set to 0. For example, when importing an event channel into a WaveMark channel, there are no marker codes or waveforms, so these are set to 0.

The **Minimum interval** field sets the minimum interval between peaks, troughs or level crossings (in the same direction) for the data to be acceptable.

The **Size** field sets the minimum peak height or trough depth. The **Level** field replaces the **Size** field when events are detected by rising and falling levels and sets the level to cross to detect an event.

When importing to a Marker or Marker-derived channel from a Level event channel, falling level transitions are given marker code 00 and rising transitions are given code 01.

The **Apply** button imports data and leaves the dialog open. **OK** imports data and closes the dialog. The **Close** button closes the dialog without importing. The **Help** button, or using the `F1` key displays this help.



Extract events from Waveform channels

You can extract events from waveform, RealWave and WaveMark channels. If the WaveMark channel has multiple traces, only the first trace is used. If the memory buffer is a Marker or is derived from a Marker, the first marker code is set to indicate the source of the data (the remaining codes are set to 0). There are six modes to extract events from waveforms:

Mode	Code	Method
Peaks	2	A peak is defined when the data rises and falls by at least the Size field value. Between each peak there is also a Trough. The peak time is estimated from the waveform around the peak using cubic spline interpolation. If the target memory buffer is a RealMark channel the first attached value is set to the peak amplitude.
Troughs	3	A trough is defined when the data falls and then rises by at least the Size field value. The trough time is estimated from the waveform around the lowest point using cubic spline interpolation. If the target memory buffer is a RealMark channel, the first attached value is set to the trough amplitude.
Peaks and Troughs	2 & 3	Peaks and troughs are defined as above. Both the peaks and troughs are saved, but with their own codes.
Data rising through level	4	The event time is set by the time where the data value rises through the nominated level. If the target memory buffer is a RealMark channel, the first attached value is set to the level. The time is estimated by cubic spline interpolation of the data.
Data falling through level	5	The event time is set by the time where the data value rises through the nominated level. If the target memory buffer is a RealMark channel, the first attached value is set to the level. The time is estimated by cubic spline interpolation of the data.
Rise and Fall through level	4 & 5	Events are generated by both rising and falling through the nominated level. The time is estimated by cubic spline interpolation of the data.

Setting the attached RealMark values and Peaks and Troughs and Rise and Fall through Level modes are new features added at Spike2 version 8.00.

If the target memory channel is of type RealMark, the first attached value is set to the Peak, Trough or Level associated with the event. Although saving the level is less useful than saving the peak or trough value, it allows you to use multiple level crossing detections in a single channel (for example, allowing you to resample a channel based on levels rather than on time intervals).

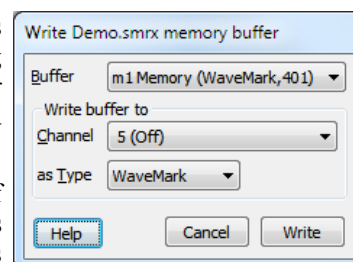
These modes use the **Minimum interval** field as the minimum separation of detected events, to filter out events caused by noise in the input waveform. Set this to 0.0 if you do not want to set a minimum period between detected events. The peak search mode looks for a peak followed by a fall of at least the **Size** field. The trough search mode looks for a minimum, followed by a rise of at least the **Size** field. The rising and falling level modes detect events when the signal crosses the **Level** in the selected direction.

To convert a waveform channel to a level event channel such that all data above a level is high and data below a level is low, create level event memory buffer with the desired initial state (low/high), then import rising edges for a level crossing, then import falling edges using the same level.

Write to channel

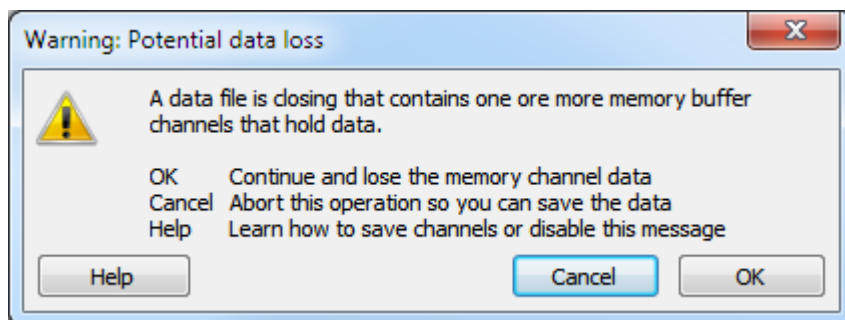
You can write a memory buffer to the data document. The **Type** field sets the format for the saved data. You can also **Append** the data to an existing channel. In **Append** mode, the data in the memory buffer must all occur later than data in the target channel. For modes other than **Append**, if you select a channel number that is already in use, you will be warned.

This command saves the entire contents of the memory buffer, regardless of any marker filter or channel process that may be attached to the channel. This is different from the **Save channel** command, which writes the data as displayed, including the effects of marker filters and channel processes.



Memory buffer channel data loss warning

This dialog is displayed when a data file is about to close and this would cause memory buffer data to be lost.



You can disable this warning, or limit when it occurs from the Edit menu Preferences in the General tab with the Warn if file close would lose memory channel data option.

If you get this message and want to preserve the data, click on **Cancel** to stop the file close process. You can then save the channels to unused channels in the data file. This can be done from the Analysis menu. There are two commands you can use:

- | | |
|---------------------------------------|--|
| Memory Buffer Write to Channel | This allows you to write a nominated memory buffer to an unused channel or to overwrite or append data to an existing channel. You can also choose the type of the channel you create. |
| Save Channel | This is a simpler command that allows you to copy any existing channel (including memory buffer channels) to an unused channel in the file. |

If you find yourself in the situation of having no available spare channels you can follow the following procedure:

1. Use the File menu **Export As** command to create a copy of the current file as a data file.
2. Set the file type to Data File (you can choose between 64-bit `smrx` and 32-bit `smr`). Set a suitable file name and location and click the Save button.
3. In the Export dialog, click **Add** (to select **All Channels** and the full time range in the file). Also click the **Channels** button and set sufficient extra channels to store your data. Clear the check in the **Time shift data** to that the first exported range starts at 0 seconds box unless this is the effect you want.
4. Click **Export** to create a copy of the original file with the memory channels saved as permanent channel.

If you want more control over the channel numbers assigned to the memory channels you can use the `ChanSave()` script command which allows you to specify exactly what you want and lets you move data between files (even files with differing time resolutions). However, you will require some expertise with the script language to use it.

Virtual Channels

Virtual channels hold RealWave data derived by a user-supplied expression from waveform, event and RealWave channels and built-in function generators. No data is stored; the channels are calculated each time you use them. You can match the sample interval and data alignment to an existing channel, or type in your own settings. Channel sample intervals and alignments are matched by cubic splining the source waveforms, linear or cubic interpolation of RealMark data and by smoothing event rates. The script language equivalent of this command is `VirtualChan()`. You can save a virtual channel to disk (to remove the calculation time penalty) with the Analysis menu **Save Channel** command.

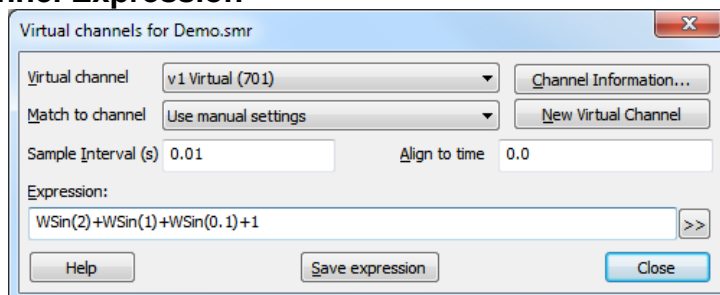
You can use virtual channels to do channel arithmetic (for example sums and differences of channels), to convert event channels into waveforms proportional to the event rate, to linearise non-linear transducers, to perform frequency analysis and to generate waveforms. This is a very powerful feature with a huge range of applications.

Because the data is calculated every time it is required, using virtual channels can be slow. If this is a problem,

you can consider saving the virtual channel to a permanent channel using the Analysis menu **Save channel** command. You can then delete the virtual channel and use the saved channel, which will be as quick to use as any other RealWave data channel.

Create New Channel, Edit Channel Expression

These commands are selected from the Analysis menu **Virtual Channels** command; they create a new virtual channel and edit the settings of existing virtual channels. Both commands open the Virtual channel dialog. The **Create New Channel** command makes a new, empty virtual channel and selects it before opening the dialog.



Virtual channel

Use this field to select a channel when you have more than one virtual channel.

Channel Information

This button is a short cut to the View menu **Channel Information** dialog; you can use that dialog to set the channel title and units.

New Virtual Channel

Click this button to create a new virtual channel.

Match to channel

You can select an existing waveform-based channel (but not a virtual channel) from which to copy the sample interval and data alignment. Alternatively, you can select **Use manual settings** and type in the interval and alignment yourself.

Sample Interval

This field holds the sample interval between data points in the virtual channel in seconds. You can edit the sample interval if you select **Manual Settings** in the **Match to channel** field. This field accepts expressions; for example, to set 27 Hz you can type $1/27$.

Align to time

This field sets the time of a data point in the virtual channel. The time of any point and the sample interval completely defines all the sample times for the channel. You can edit the alignment if you select **Manual Settings** in the **Match to channel** field.

Expression

This field holds an expression that defines the virtual channel. Expressions are composed of scalars, vectors, operators, channel functions, spectral functions, event kernel functions, waveform generation and mathematical functions

A scalar is a number, such as 4.6 or $\text{Sqrt}(2)$. A vector is a list of data values that are derived from a channel or generated by a waveform function. A function can be applied to a vector or a scalar to yield a result that is a vector or a scalar. An operator combines vector and scalars, for example $1 + 2$ combines the scalars 1 and 2 to generate the scalar 3. $\text{Ch}(1)+1$ combines the vector $\text{Ch}(1)$ with the scalar 1 to give a vector that is the waveform from channel 1 increased by 1.

Example expressions

If channels 1 to 3 hold waveforms, then $\text{Ch}(2) - 2*\text{Ch}(1)$ displays the difference between channel 2 and twice channel 1.

$\text{Sqrt}(\text{Sqr}(\text{Ch}(1))+\text{Sqr}(\text{Ch}(2))+\text{Sqr}(\text{Ch}(3)))$ displays the square root of the sum of squares of three channels. You could use this to display the magnitude of the resultant of three perpendicular forces.

`Sqr(Ch(1))` is the same as `Ch(1)*Ch(1)`, but `Sqr()` is faster. To generate a polynomial function of the input it is much quicker to use `Poly()` than to use `Ch()`, `Squ()`, `Cub()` and so on to generate a power series.

Error messages

The expression is checked as you type it. If the expression is incorrect, an explanatory message appears to the right of **Expression:** above the expression entry field.

>>

Click this button to open the Build Expression drop-down list, which helps you to construct expressions interactively without needing to know the virtual channel expression syntax.

Save expression

There is a list of saved expressions in the Build Expression drop-down list. Valid expressions are added to the list when you click the **Save expression** button, when you change to a different virtual channel and when you close the dialog with the **Close** button. The expressions are stored in the system registry.

Operators

Arithmetic operators

You can use the four standard arithmetic operators plus (+), minus (-), multiply (*) and divide (/) and comparison operators together with numbers, round brackets and some mathematic and channel functions. The result of combining a vector and scalar with an operator is a vector, for example, the expression `Ch(1)+1` is a vector, being the data points of channel 1 with 1.0 added to each of them.

Dividing by a scalar value of zero is an error. Dividing by a vector holding zeros is not an error and generates special floating-point numbers for positive and negative infinities. These can cause problems in subsequent calculations (and they are difficult to display).

Comparison operators

You can also use comparison operators less than (<), less than or equal (<=), equal (=), not equal (<>), greater than or equal (>=) and greater than (>); the result of these is 1.0 if the comparison is true and 0.0 if it is false. In the expression `a op b`, where `a` and `b` are vectors and/or scalars and `op` is a comparison operator, if either `a` or `b` is a vector, the result is a vector with value 1.0 at points where the comparison is true and 0.0 where it is false. For example `Ch(1)>1` has the value 1.0 where channel 1 is greater than 1.0 and 0.0 elsewhere. `Ch(1)<>Ch(2)` is 1 wherever channels 1 and 2 are not the same. Be cautious using `<>` (not equals) and `=` (equals) as we are dealing with floating point numbers and exact equality may be compromised by arithmetic rounding.

Operator precedence

Multiply and divide have a higher precedence than all the other operators which all have the same precedence. You can use round brackets to force other evaluation orders. Apart from that, evaluation is from left to right, so `a>b*c+d` is interpreted as `(a>(b*c))+d`.

Channel functions

The following functions create a vector at the sample interval and alignment set for the virtual channel from an existing data channel (but not a virtual channel unless it has a lower channel number than the channel in use).

<code>Ch(n)</code>	<code>n</code> is a waveform, <code>RealWave</code> , <code>WaveMark</code> or a level event channel. Copy channel <code>n</code> data. Level event channels copy as 1 when the level is high and 0 when low. <code>WaveMark</code> channels copy the waveform from the first trace only and missing data is set to 0; script users can access the other traces with the <code>NextTime()</code> and <code>LastTime()</code> commands.
<code>If(n,g)</code>	<code>n</code> is an event channel to convert to a waveform by linear interpolation of the instantaneous frequency. <code>g</code> is the maximum gap to interpolate across, in seconds. Omit <code>g</code> or set it to 0 for no limit to the gap.
<code>Ifc(n,g)</code>	The same as <code>If()</code> except that cubic spline interpolation is used.

$Rm(n, g, i)$	n is a RealMark channel to convert to a waveform by linear interpolation of the real data values, g is the maximum gap to interpolate over in seconds, and i is the real data item to interpolate (the first item is 0). Set g to 0 for no maximum gap. You can omit i or both i and g . Omitted values are treated as 0.
$Rmc(n, g, i)$	The same as $Rm()$ except that cubic spline interpolation is used.

Since version 7.09a, you can use channel specifiers like $m1$, $v2$, $1a$ in places where only a channel number was previously acceptable.

Use of virtual channel numbers in an expression

To prevent recursive references, you can only use a virtual channel in an expression if it refers to a lower-numbered virtual channel than the channel being defined. Although this can sometimes make things easier to visualise and allows common sub-expressions to be used by other virtual channels, this does not save you any calculation time (any may even be slower) than writing out the full expression. This is because the virtual channel is calculated whenever it is needed; it is not cached. A virtual channel may not depend on a duplicate of a virtual channel.

Spectral functions

The following functions all calculate the power spectrum of an input channel and generate a waveform illustrating how some feature of the power changes with time. These functions are particularly useful in EEG and EMG analysis. The power spectrum is calculated using the Fast Fourier Transform (FFT), applied many times to span the data. These functions can take some time to calculate. If you find that these operations are slow, consider saving the virtual channel as a real channel so that you need only suffer the slow operation once. Arguments common to all commands are:

n	The input waveform or RealWave channel. The maximum frequency for use in bands available to any of the commands is half the sample rate of this channel. Missing data values (gaps) in the input channel are treated as zero values.
f	The desired frequency resolution of the output. For a resolution of f Hz, the FFT used to calculate the power must span at least $1/f$ seconds of the input data. So 0.1 Hz resolution implies an FFT spanning 10 or more seconds of input data. The FFT we implement uses a power of 2 data points, which constrains the available resolutions; we round f down to the nearest available value. There is a trade-off between frequency and timing resolution. If you set f to 0, the FFT size is 256 points.
l	The low edge of a frequency range, in Hz. If this is an optional argument and you omit it, 0 is used.
h	The high edge of a frequency range, in Hz. if this is an optional argument and you omit it, half the sampling rate of channel n is used.

The available functions are:

$Pw(n, f, l, h)$	Calculate the power in a frequency band from l to h Hz. There is a dialog to help you build the expression.
$Pw(n, f, l, h, r, s)$	Calculate the power in the frequency band from l to h Hz divided by the power in the band from r to s Hz. If s is omitted, the band extends from r to half the sample rate of the input channel.
$SpE(n, f, p, l, h)$	Calculate the frequency at which p percent of the power in the band from l to h Hz lies below. If l is omitted, 0 Hz is used, if h is omitted, half the sample rate of the input is used. There is a dialog to help you build the expression.
$MF(n, f, l, h)$	Calculate the mean frequency in the band from l to h . If l is omitted, 0 Hz is used, if h is omitted, half the sample rate of the input is used. The mean frequency is sum of products of frequency times power divided by the sum of the power. There is a dialog to help you build the expression.
$DF(n, f, s, l, h)$	Calculate the Dominant Frequency. We search the band from l to h Hz for the frequency region with the maximum power. s sets the distance in Hz to smooth the data either size of each point (0 for no smoothing). There is a dialog to help you to build the expression.

Technical details

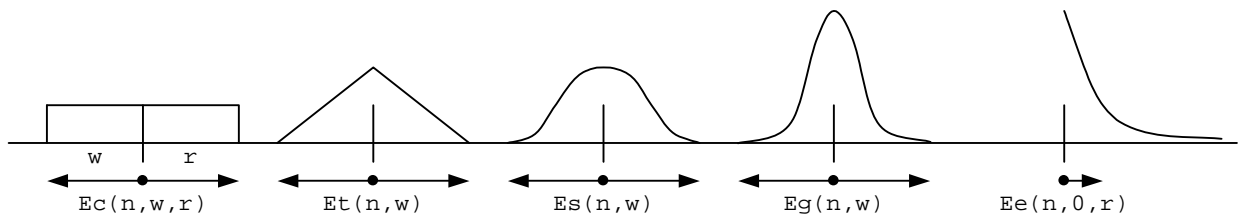
Power is calculated using the Fast Fourier Transform (FFT) with a raised cosine window. There are three regimes used to calculate the power, these depend on the ratio of the width of the FFT (being the FFT size times the sampling interval of the input waveform) relative to the interval between the output data points:

Ratio	Method
>8	We calculate the power spectrum with a single FFT at a time interval of the FFT width/8 and then interpolate the results to the output. This prevents excessive calculations when the user sets a high output sample rate. In this case, 3/4 of the data points are shared in common between two consecutive transforms, so the change between them is small.
8 to 2	We calculate the power spectrum with a single FFT centred on each output point.
<2	We calculate the power spectrum with multiple FFTs. These span a space from 1/2 an output interval plus 1/4 of the FFT width before each output point to the same distance after each point and the FFTs overlap by at least 50%.

This sequence should achieve a smooth transition between different regimes as the output sampling rate changes.

Event kernel functions

The $E_c(n, w, r)$, $E_t(n, w, r)$, $E_s(n, w, r)$, $E_g(n, w, r)$ and $E_e(n, w, r)$ functions convert event channel n into a virtual wave and can be used anywhere in an expression that you can use the $Ch()$ function. They replace each event by a kernel (shape), centred on the event time. For an event at time t , the kernel extends from $t-w$ to $t+r$ seconds except for $E_e()$, which extends from $t-8w$ to $t+8r$ seconds. Normally you will omit r , in which case the shapes are symmetrical with r set equal to w . The resulting waveform is the sum of the kernels for all the events. The area of each kernel is unity, so the area under the waveform between any two times is the number of events in that time interval.



- $E_c(n, w, r)$ The replacement shape is a rectangle, think Event Count, running from w to the left to r to the right.
- $E_t(n, w, r)$ The replacement shape is a Triangle running from w to the left to r to the right.
- $E_s(n, w, r)$ The replacement shape is a Sinusoid running from w to r to the right.
- $E_g(n, w, r)$ The replacement shape is a Gaussian with a sigma (standard deviation) of $w/4$ to the left and $r/4$ to the right.
- $E_e(n, w, r)$ The replacement is an exponential function with a time constant of w to the left of each event and r to the right. The kernel extends to 8 times the time constant in each direction.

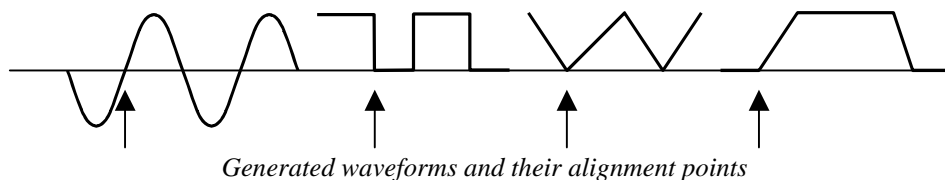
The $E_c()$ function simply counts the events in the time range. This is the fastest analysis method, but produces the most jagged output. The $E_t()$ function weights the events with a triangle function. This is slower than $E_c()$, but much faster than $E_s()$ and $E_g()$. The $E_s()$ function weights the events with a raised cosine. The $E_g()$ function weights the events with a Gaussian curve extending to 4 sigmas.

The $E_e()$ function is rather different from the others as it is not suitable for use as a smoothing function and is more likely to be used in a single-sided form. It weights the events with an exponent $\exp(-t/r)$ to the right and $\exp(-t/w)$ to the left (t stands for the time difference between the point and the time). The exponent extends to $8w$ to the left and to $8r$ to the right.

If none of these conversions are suitable, you can define your own weighting functions and create a new (not virtual) channel with the `EventToWaveForm()` script command.

Waveform generation

These functions generate waveforms without the need for an input channel. All these functions produce output from time 0 to the end of the file. All the arguments are in units of seconds, except the sine wave frequency, which is in Hz. All these waveforms have unit amplitude. You can use the standard maths operators $*$ / $+$ and $-$ to scale and shift the output to different values.



If you attempt to create a cyclic waveform with a frequency above half the channel sample rate, no output will be generated. The a argument sets the time alignment; if omitted the value 0 is used. You can generate the following outputs:

- WSin**(f, a) Sine wave of frequency f Hz aligned so that phase 0 (the point where the rising sinusoid crosses 0) is at time a seconds. The amplitude of the output runs from -1.0 to +1.0. The sinusoid is most accurate at the start of a file; the accuracy falls off as the number of cycles increases (this is not usually noticeable).
- WSqu**(l, h, a) Square wave with low period l seconds and high period h seconds aligned so that a low period starts at time a seconds. Both l and h must be greater than 0 seconds. The output level of the low section is 0, the output level of the high section is 1.
- WTri**(r, f, a) Triangle wave with a rise time of r seconds and a fall time of f seconds aligned so that a rise starts at time a seconds. Either of r or f may be zero, but not both. The triangle output level is from 0 to 1.
- WEnv**(r, h, f, a) Envelope with a rise time of r seconds, a hold time of h seconds, a fall time of f seconds with the rise starting at time a seconds. The output waveform is 0 before time a and after time $a+r+h+f$ and is 1 during the hold time. At least one of r, h or f must be non-zero.
- WPoly**(f, t, r, L) Polynomial in time from f to t seconds and 0 before f and from t . The value at time t is a function of $(t-r)$ where r is a reference time. L is a list of 1 to 6 coefficients. **WPoly**($f, t, r, a, b, c, d, e, f$) is:
- $$y(t) = a + b*(t-r) + c*(t-r)^2 + d*(t-r)^3 + e*(t-r)^4 + f*(t-r)^5$$
- WT**(s, e) Ramp from time s up to time e of value at time t of $(t-s)$ with value 0 before s and from e . Omit e to run to the end of the file, omit both e and s to start from the beginning.

Mathematical functions

The mathematical functions all have the form **Func**(x), where x can be either a scalar or a vector and **Func**() is the operation. If x is a vector, the result is a vector with the function applied to each value otherwise the result is a scalar.

- Abs**(x) The absolute value of x (negative values are replaced by positive values of the same size). Use this to rectify a vector.
- Hwr**(x) Half wave rectify x . Negative values are replaced by zeros. This is faster than **Max**($x, 0$).
- Sqr**(x) This calculates the square of x . **Sqr**(x) is the same as $x*x$, but is faster, particularly when x is a vector.
- Cub**(x) This calculates the cube of x . **Cub**(x) is the same as $x*x*x$ but is much faster, particularly when x is a vector.
- Sqrt**(x) This calculates the square root of x . When x is a vector, negative values are set to 0. If x is a scalar, negative values cause an error.
- Ln**(x) Natural logarithm of x . Negative or zero x values generate -100 when x is a vector and an error when x is a number.
- Exp**(x) Exponential of x . If the result overflows, this is an error when x is a number and sets the

	largest allowed result when x is an array.
<code>Max(x,y)</code>	The maximum of x and y . This can be used to set a lower limit, for example <code>Max(Ch(1),1)</code> is a vector with minimum value 1.
<code>Min(x,y)</code>	The minimum of x and y . This can be used to set an upper limit, for example <code>Min(Ch(1),Ch(2))</code> can be thought of as the value of channel 1 with the maximum value limited by the value of channel 2 (or vice versa).
<code>Sin(x)</code>	Calculates the sine of x (x is in radians). If x is in degrees, divide by 57.295779513 to convert to radians.
<code>Cos(x)</code>	Calculates the cosine of x (x is in radians).
<code>Tan(x)</code>	Calculates the tangent of x (x is in radians). The result can be infinite.
<code>ATan(x)</code>	The arc tangent of x . The result is in radians in the range $-\pi/2$ to $\pi/2$.
<code>ATan(s,c)</code>	Both s and c must be vectors or both must be scalars. The result is the arc tangent of s/c with the quadrant set by the signs of s and c . The result is in the range $-\pi$ to π .
<code>Poly(x,L)</code>	Replace x with a polynomial in x of order 1 to 5; L is a list of 1 to 6 coefficients. Use this to apply a non-linear calibration to a signal. For example: <code>Poly(x,a,b,c,d) = a + b*x + c*x*x + d*x*x*x</code>

You can insert spaces between operators and numbers and between round brackets and the items within them. You may not insert spaces between a function name and the opening bracket that follows it.

The multiply and divide operators have higher precedence than add and subtract, so `1/2+3*4` is 12.5. You can use brackets to force other evaluation orders, for example `1/(2+3)*4` is 0.8. Apart from that, evaluation is from left to right.

Dividing by a scalar value of zero is an error. Dividing by a vector holding zeros is not an error and generates special floating-point numbers for positive and negative infinities. These can cause problems in subsequent calculations (and they are difficult to display).

Build expression

There is no need to remember all the expression functions; just click the `>>` button and choose from a list of possible items to add. All the commands in this section generate a text string that is inserted at the caret position (replacing any selection) in the **Expression** field. Simple command insert the text immediately; more complex commands open a dialog to build the expression. Commands that open dialogs display the expression at the lower left corner. You can choose from:

Waveform from channel	These commands create a waveform from an existing channel.
Spectral functions	These commands create waveforms based on the spectral content of a channel. You can choose from Power in band or ratio of bands, Spectral edge, Mean frequency and Dominant frequency.
Generate waveform	Create a waveform independently of any channel, for example a sinusoid or a triangle wave or an envelope function.
Rectify and Absolute value	These commands insert the <code>Abs()</code> , <code>Hwr()</code> , <code>Max()</code> and <code>Min()</code> commands that rectify and limit values.
Mathematical functions	These commands insert mathematical functions (<code>Sqrt()</code> , <code>Sin()</code> , <code>Cos()</code> etc) and also the <code>Poly()</code> command which generates a polynomial of a vector or scalar.
Mathematical operators	Select one of these to insert the operator to replace the selection..
Previous virtual channel expressions	This lists expressions that you have used previously; the most recently used is listed first. Valid expressions are added to the list when you click Save expression , when you change to a different virtual channel and when you close the dialog with Close . The expressions are stored in the system registry.

Apply polynomial to data

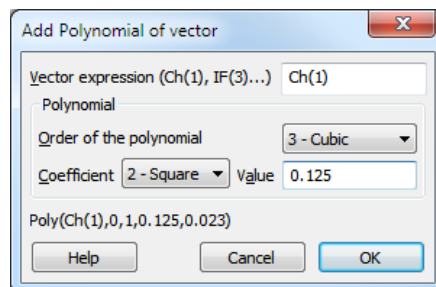
This dialog is opened from the Virtual channel expression field. The Vector expression field is set to whatever was selected when this dialog opened, or to `Ch(1)` if nothing was selected. This field is not tested for validity. Each element x of the vector is replaced by a polynomial in x . You can set the order of the polynomial (order means the highest power of x used) in the range 1 to 5. The command is:

```
Poly(x, a0, a1, a2, a3, a4, a5)
```

where x is the vector expression (you can use a scalar, but this is not very useful) and the a_0 to a_5 are the coefficients of the polynomial (you must supply these). This expression generates the quintic:

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5$$

For lower order polynomials, omit coefficients from the right. For example, for a quartic (fourth order polynomial), omit a_5 , for a cubic omit a_5 and a_4 . To set coefficient values in the dialog, use the Coefficient field to select the required coefficient and then set its value.

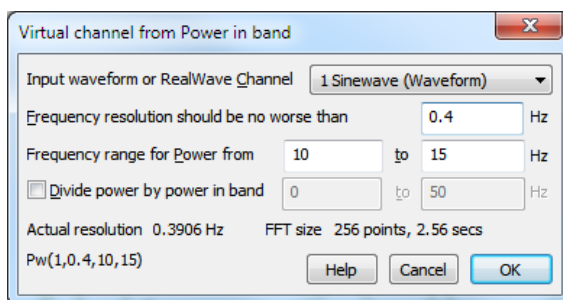


Power in Band

This dialog (accessed from the Virtual Channel build expression dialog) generates a waveform channel representing either the power in a frequency band, or the ratio of power in a band to the power in another band. The fields are:

Input waveform or RealWave Channel

Use this to select a suitable input channel to be analysed. The maximum values you can set for the frequency band is half the sample rate of this channel. If you change the channel, the frequency resolution field and the frequency range are reset to default values.

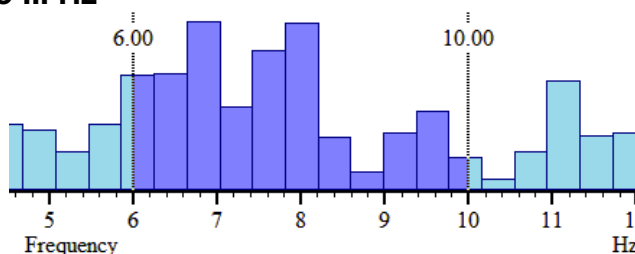


Frequency resolution should be no worse than ... Hz

When you open the dialog or change the channel, this value is set to the frequency resolution you would get for a FFT size of 256 points (which is usually a sensible value). Follow this link for more information about the frequency resolution. The actual frequency resolution and FFT size that will be used to achieve it are displayed in the dialog.

Frequency range for Power from ... to ... Hz

These fields set a frequency range where the power is to be calculated. The lower frequency must be less than half the sample rate of the source channel. The upper frequency must be greater than the lower. The results of the power calculation are in bins of fixed frequency width so it is unlikely that the frequencies you set will match the edges of the power bands. The power is calculated starting at the lower frequency and extending up to the the upper one. If a band starts or ends with a fraction of a bin, then that fraction of the power in the bin is included.



Divide power by power in band from ... to ... Hz

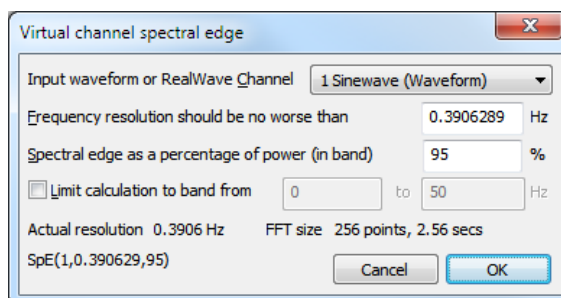
If you check the box, the result is not power, but is the ratio of the power in two bands. Usually, the second band is set from 0 to half the sampling rate, giving a measure of the proportion of the total power, but you can set any frequency range you like as long as the lower frequency is less than half the sample rate. If the band

used for division does not completely overlap the first band there is the possibility of division by zero. If this occurs, the result is set to 1000000 (rather than setting infinity, which is difficult to draw).

If you do not check the box, the output waveform is calibrated in power such that the output is the mean square of the data in the frequency band. That is, if the input data were a sinusoid of amplitude A units of a frequency in the band, the output would be $A^2/2$ units squared. Put another way, if the frequency band was set to include all frequencies, the output would be more or less that same as the input channel squared and then smoothed over a time of the FFT size.

Spectral Edge

This dialog (accessed from the Virtual Channel build expression dialog) generates a waveform channel holding the frequency at which the sum of the power starting at 0 (or the low edge of a defined band) is a used defined percentage of the total power (or the total power in a defined band). The fields are:



Input waveform or RealWave Channel

Use this to select a suitable input channel to be analysed.

The maximum values you can set for the frequency band is half the sample rate of this channel. If you change the channel, the frequency resolution field and the frequency range are reset to default values.

Frequency resolution should be no worse than ... Hz

When you open the dialog or change the channel, this value is set to the frequency resolution you would get for a FFT size of 256 points (which is usually a sensible value). Follow this link for more information about the frequency resolution. The actual frequency resolution and FFT size that will be used to achieve it are displayed in the dialog.

Spectral edge as a percentage of power (in band) ... %

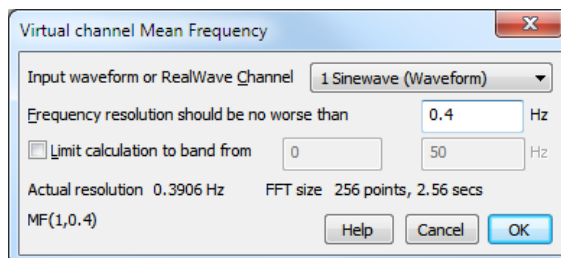
You set the proportion of the signal as a percentage in the range 0 to 100. If you want the Median frequency as used in some EMG work, set the percentage to 50.

Limit calculation to band from ... to ... Hz

If you check the box, you can set limits for the band of frequencies that are used in the calculation. The range of bins used is inclusive from the nearest bin in the power spectrum to the low frequency up to the nearest bin to the high frequency.

Mean frequency

This dialog (accessed from the Virtual Channel build expression dialog) generates a waveform channel holding the mean frequency, defined as the sum of the product of Power and frequency divided by the sum of the power. You can limit the calculation to a range of frequencies. The fields are:



Input waveform or RealWave Channel

Use this to select a suitable input channel to be analysed. The maximum values you can set for the frequency band is half the sample rate of this channel. If you change the channel, the frequency resolution field and the frequency range are reset to default values.

Frequency resolution should be no worse than ... Hz

When you open the dialog or change the channel, this value is set to the frequency resolution you would get for a FFT size of 256 points (which is usually a sensible value). Follow this link for more information about

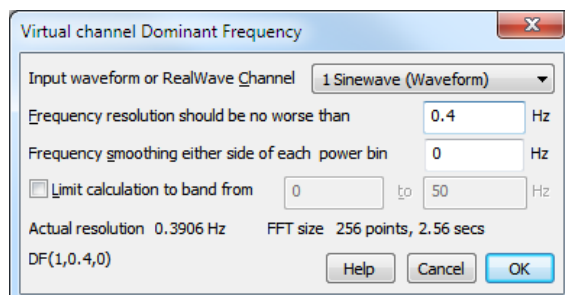
the frequency resolution. The actual frequency resolution and FFT size that will be used to achieve it are displayed in the dialog.

Limit calculation to band from ... to ... Hz

If you check the box, you can set limits for the band of frequencies that are used in the calculation of mean frequency. You could use this to exclude a DC offset from the calculation.

Dominant frequency

This dialog (accessed from the Virtual Channel build expression dialog) generates a waveform channel holding the frequency at which the most power was found. You can limit the search to a frequency range. The fields are:



Input waveform or RealWave Channel

Use this to select a suitable input channel to be analysed. The maximum values you can set for the frequency band is half the sample rate of this channel. If you change the channel, the frequency resolution field and the frequency range are reset to default values.

Frequency resolution should be no worse than ... Hz

When you open the dialog or change the channel, this value is set to the frequency resolution you would get for a FFT size of 256 points (which is usually a sensible value). Follow this link for more information about the frequency resolution. The actual frequency resolution and FFT size that will be used to achieve it are displayed in the dialog.

Frequency smoothing either side of each power bin ... %

If you leave this value as 0, the result is the power bin with the largest power and the result is quantised to the frequency resolution. The value you set here is divided by the frequency smoothing to give the number of extra bins either side of each searched bin to include when looking for the maximum. If the extra bins would extend before 0 Hz or after the bin corresponding to half the sampling rate, these non-existing bins are presumed to hold 0 power. When a maximum is found, the result is the weighted mean frequency in the range of bins.

Limit calculation to band from ... to ... Hz

If you check the box, you can set limits for the band of frequencies that are used as the centre of the search for the dominant frequency. The actual result can be outside this range if the frequency smoothing is set and a large power peak is just outside the range.

Spectral frequency resolution

The three virtual channel functions that generate spectral functions all have a frequency resolution parameter. To best use this you need to understand how the spectral functions are calculated. Spike2 converts from a waveform (the time domain) to power at a given frequency (the frequency domain) using a radix 2 Fast Fourier Transform (FFT). The analysis we do takes a sequence of n input data points and generates $n/2+1$ output points that describe the power, spanning the frequency range 0 to half the waveform sampling rate. The output points are equally spaced in frequency.

The frequency spacing of the output points is thus the input waveform sample rate divided by the number of points, n in the transform. However, the sample rate is also the reciprocal of the sample interval, and n times the sample interval is the time duration of the input n points:

$$\text{Frequency resolution } \epsilon = \text{sample rate} / n = 1 / (n * \text{sample interval}) = 1 / (\text{input duration})$$

When you specify the frequency resolution, you are also specifying the time duration of the FFT transform. This means that if you want the power at time t , with a frequency resolution of ϵ Hz, we must transform the

data from time $t-1/(2*\epsilon)$ to $t+1/(2*\epsilon)$ seconds. That is, the more accurately you want to determine frequency, the less accurately the position in time where the frequency occurred can be known.

The details

The FFT we use does not accept arbitrary lengths of input data; it works with powers of 2 data points. If the number of samples set by a duration of $1/\epsilon$ seconds is not a power of two (as is generally the case), it is rounded up to the next power of two. The value of ϵ used will always be less than or equal to the value that you request. The resolution ϵ , the FFT size used and the time period that the FFT spans are displayed in the dialog so you can make sure that the settings makes sense for your application.

Each waveform section that is transformed is first multiplied by a Hanning window. This weights the input data so that the values nearer to the centre get more weight in the result than those at the edges; it also prevents artefacts in the result.

If the sample interval for the virtual channel is more than half the FFT size in seconds, the result for a particular point is calculated by using overlapped transforms (the overlap factor is at least half the FFT width) of the input data from half an output sample interval before the point to half a sample interval after.

If the sample interval for the virtual channel is half the FFT size down to 1/8 the FFT size, each output point is calculated from one transform with the data centred on the output point.

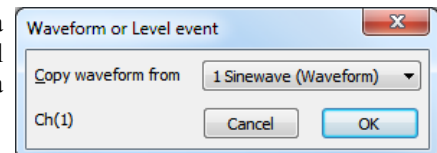
If the sample interval for the virtual channel is less than 1/8 the FFT size, we calculate the power spectrum for an interval of 1/8 of the FFT size, then interpolate the values.

Waveform from Channel

The commands in the Waveform from Channel section of the Virtual channel dialog expression field generate waveforms directly from data channels. The commands all open a dialog:

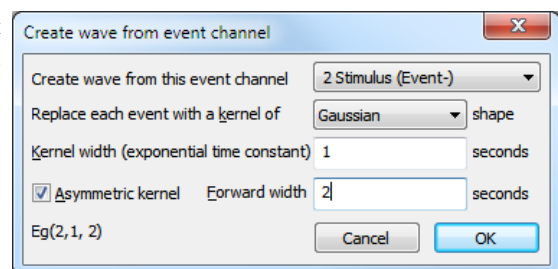
Copy Waveform or Level event

This dialog builds the `Ch()` command, allowing you to copy data from a waveform, RealWave, WaveMark (first trace only) and Level event channel. If you select a level event channel, the output is 0 for a low level and 1 for a high level.



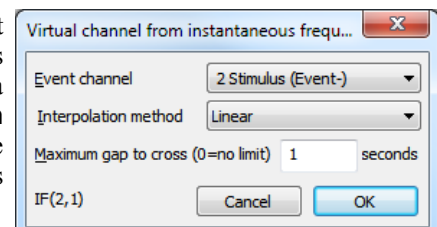
Event channel using kernel

This dialog lets you choose between the various Event kernel functions: Rectangular, Triangular, Raised Sinusoid, Gaussian and Exponential. Normally you will leave the Asymmetric kernel box unchecked so that the shape that replaces each event is symmetric about the event time. The shape is scaled so that the area under it is unity (time measured in seconds).



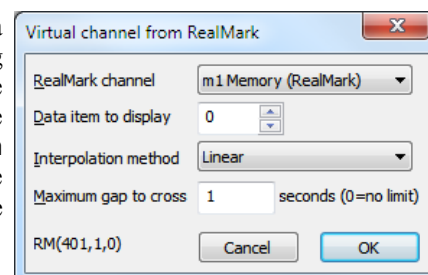
Event instantaneous frequency

This dialog generates the `IF()` and `IFC()` commands, which convert a channel of event times into a waveform of the channel instantaneous frequency. The frequency points at the events can be linked by a straight line (linear) or by a cubic spline. If you set the Maximum gap to cross field to a non-zero value, this will generate gaps in the waveform if the gap between two instantaneous frequency values exceeds the gap.



RealMark data item

This dialog generates a waveform from the data values attached to a RealMark channel. Each RealMark data item has a list of floating point values for data items associated with it, so you must choose which one to display. The points can be joined by a straight line (linear interpolation) or by a cubic spline. If you set the **Maximum gap to cross field** to a non-zero value, this will generate gaps in the waveform if the gap between two RealMark items values exceeds the gap.

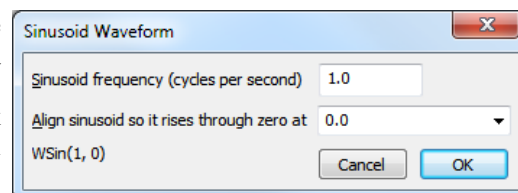


Generate Waveform

The commands in the **Generate Waveform** section of the Virtual channel dialog expression field generate waveforms without any reference to data channels. Note that these commands generate data that can only exist in the time span from 0 to the length of the data file defined by other channels. If you have a completely empty data file you will need to add a real or a memory channel with some data at the maximum time you want to generate data for. You can generate:

Sinusoid

This generates a `WSin()` command that extends for the entire length of the data file. You set the frequency in Hz and a point in time at which the sinusoid rises through zero. The generated waveform has mean value of 0 and a peak to peak amplitude of 2. The waveform frequency must be less than twice the sampling rate set for the channel.



Square wave

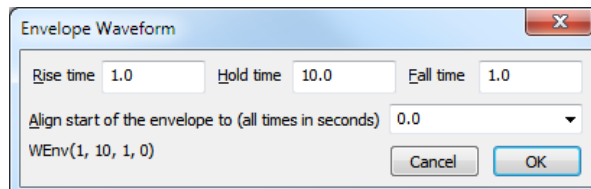
This generate a square wave that extends for the entire length of the data file with the `WSqu()` command. You define the wave by setting the low (waveform value 0) and high (waveform value 1) time periods and an alignment point where the waveform rises from zero. The period of the waveform (low plus high time) must be at least two sample periods of the virtual channel.

Triangle wave

This generates a triangle wave that extends for the entire length of the data file with the `WTri()` command. The waveform is defined by a rise time and a fall time in seconds and by a time at which a rise starts. The period of the wave must be at least two sample periods of the virtual channel.

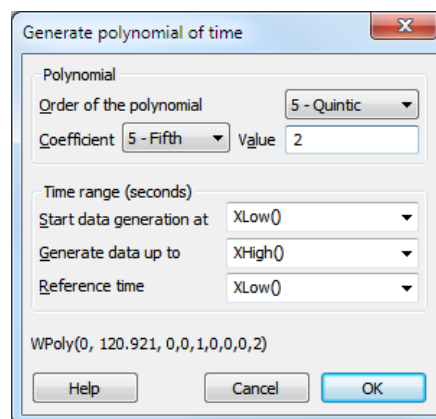
Envelope

An envelope is zero everywhere except in one region where the value rises linearly to 1, holds at 1, then falls linearly back to 0. It is implemented by the `WEnv()` command. It is defined by the start time, rise time, hold time and fall time, all in seconds.

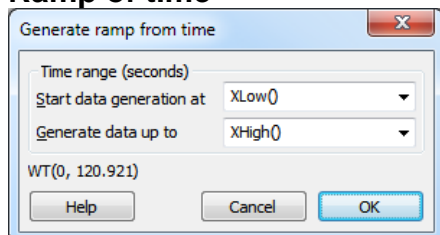


Polynomial of time

The `WPoly()` command generates polynomials in time relative to a reference time, time being measured in seconds. The polynomial has a value in a defined time range and is zero elsewhere. Such curves can be used to create complex envelopes, or to subtract out a curve generated by the interactive curve fitting routines. The order of the polynomial can be set to: Constant, Linear, Quadratic, Cubic, Quartic or Quintic, representing the highest power of the time used. The Coefficient field selects the coefficient to set in the Value field.



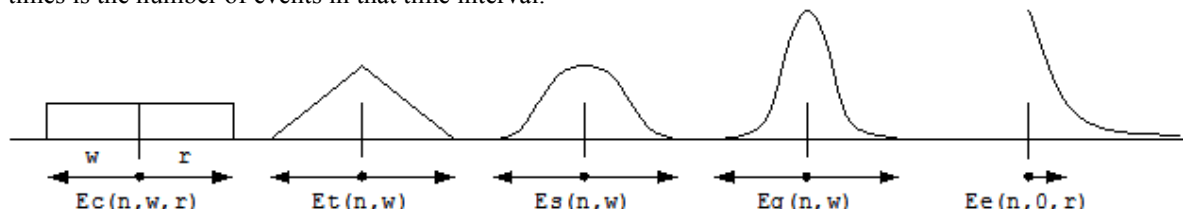
Ramp of time



The `WT()` command generates a ramp from a start time up to, but not including an end time. The data is zero outside this time range. Within the time range, the ramp value is the current time in seconds minus the start time. You will usually want to scale the ramp by multiplying it by a constant value.

The Event to Waveform functions

The `Ec(n,w,r)`, `Et(n,w,r)`, `Es(n,w,r)`, `Eg(n,w,r)` and `Ee(n,w,r)` functions convert event channel n into a virtual wave and can be used anywhere in an expression that you can use the `Ch()` function. They replace each event by a kernel (shape), centred on the event time. For an event at time t , the kernel extends from $t-w$ to $t+r$ seconds except for `Ee()`, which extends from $t-8w$ to $t+8w$ seconds. Normally you will omit r , in which case the shapes are symmetrical with r set equal to w . The resulting waveform is the sum of the kernels for all the events. The area of each kernel is unity, so the area under the waveform between any two times is the number of events in that time interval.



The `Ec()` function simply counts the events in the time range. This is the fastest analysis method, but produces the most jagged output. The `Et()` function weights the events with a triangle function. This is slower than `Ec()`, but much faster than `Es()` and `Eg()`. The `Es()` function weights the events with a raised cosine. The `Eg()` function weights the events with a Gaussian curve extending to 4 sigmas.

The `Ee()` function is rather different from the others as it is not suitable for use as a smoothing function and is more likely to be used in a single-sided form. It weights the events with an exponent $\exp(-t/r)$ to the right and $\exp(-t/w)$ to the left (t stands for the time difference between the point and the time). The exponent extends to $8w$ to the left and to $8r$ to the right.

If none of these conversions are suitable, you can define your own weighting functions and create a new (not virtual) channel with the `EventToWaveform()` script command.

Duplicate Channels

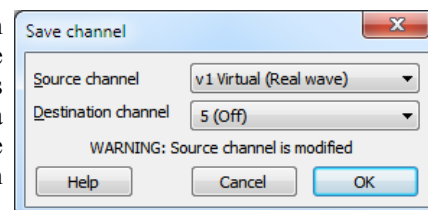
This Analysis menu command duplicates selected channels in the current time or result view. Duplicate channels share data, channel scales and comment with the parent and inherit the channel settings. Once you have duplicated a channel you can change title, display mode and y axis range independently of the original. With time view marker data, you can change the marker filter and the marker code (1-4) that is displayed.

The channel number of a duplicated channel is displayed as the original channel number plus a letter. The first duplicate of a channel gets the letter `a`, the second `b` and so on up to `z`, then `A` to `Z` are used. Duplicate

channels are deleted with the Analysis menu Delete Channel command. New duplicates get the lowest available free letter. Duplicated channels are added into the display as close as possible to the real channel they duplicate, or if there are already duplicates, as close as possible to the last duplicate. If the closest channel is part of a group, the new channel is added after or before the group, depending on the channel sort order set in the Edit menu Preferences.

Save channel

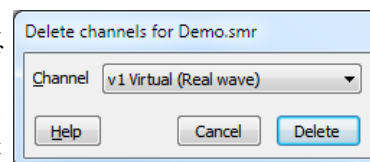
This copies the channel selected with the Source channel field to an unused channel on disk set by the Destination channel field. The channel is saved as displayed; any changes due to channel processes or marker filters are preserved in the output. The dialog displays a warning if this is the case. This command is not the same as the memory buffer Write to channel command, which saves the data in the memory buffers and allows you to set the destination channel type.



The ChanSave() script command also allows you to save channels, but gives you more control, including the ability to time shift data and to write data to a different file.

Delete channel

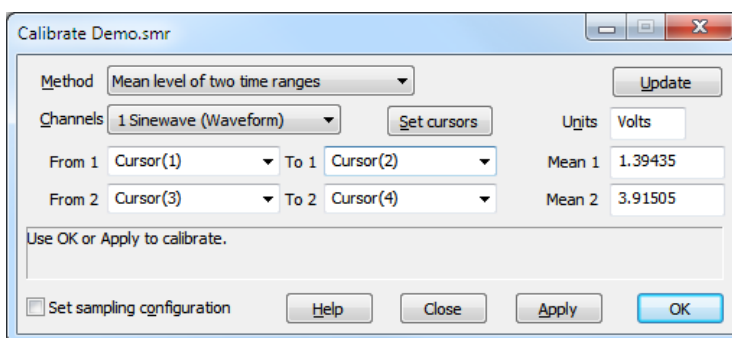
This removes a channel from a Spike2 data file permanently or deletes a channel created with New Buffer from the Memory Buffer command. If you delete a channel with duplicates, the duplicates are also deleted. Deleting a duplicated channel removes the duplicate only. You can also delete channels (except the last one) from XY views and duplicated result view channels. It is also possible to delete items (not the entire channel) from a memory buffer with the Memory Buffer Delete Items command.



Calibrate

If a waveform, WaveMark or RealWave channel has sections with known amplitudes, offsets, slopes or areas, you can calibrate it. Spike2 supports a wide variety of calibration methods for these channels. You can calibrate a single channel, or all selected channels (as long as they have the same calibration values). If you make a mistake, you can Undo the calibration changes. You can also calibrate from a script.

To calibrate your data, click on a time window and open the calibrate dialog. If there are suitable channels selected, the Channels field shows them, otherwise you must choose one. Once the dialog is open you can select channels in the time window; these are added to the selected list in the dialog.



Next, choose a calibration method. The dialog contents change depending on the method. All methods require you to select data areas with known values; this is most easily done with cursors. Click Set cursors to position appropriate cursors in the time window and dialog. You can also type in the times and use expressions like $(C1+C2)/2$. If the method requires two time ranges they must not overlap.

When you change the method or the channel or click the Update button, Spike2 collects the current calibration values from the (first) channel in the Channels field.

Once you have selected your data you must type in the calibration values, and also set the units of the data. A text box displays Use OK or Apply to calibrate or an explanation of any problem that prevents calibration. Click Apply to calibrate and leave the dialog open or OK to calibrate and close the dialog.

If you check the Set sampling configuration box, any calibration changes are passed through to the same

channel number in the sampling configuration. If you calibrate with a data file that you are sampling, or that you have just sampled and not yet saved, this box is checked automatically when the dialog opens.

What calibration does

Waveforms stored on disk as 16-bit integers (range -32768 to 32767) are scaled into user units by a *scale* and *offset*: $\text{User units} = 16\text{-bit integer} * \text{scale} / 6553.6 + \text{offset}$

The factor 6553.6 is present so that if the input data range spans -5 to +5 Volts (as is usually the case with a CED 1401 interface), the relationship is:

$$\text{User units} = \text{input in Volts} * \text{scale} + \text{offset}$$

The *scale* and *offset* are the same values as you set in the sampling configuration dialog or in the channel information dialog in a time window. Calibration changes the *scale* and *offset* so that the displayed data matches your user units.

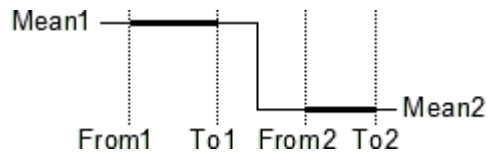
RealWave data is stored as floating point numbers on disk. Calibration rewrites this data and so may take noticeable time when working on very large data files. You cannot calibrate a RealWave data if any channel process is attached. You cannot calibrate any channel with an attached process that has changed the channel scale or offset value.

Calibration methods

The Method field of the calibration dialog sets how the calibration is performed:

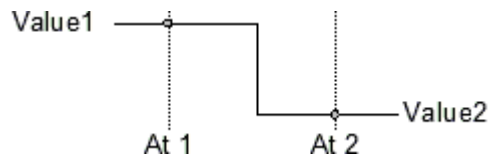
Mean level of two time ranges

You define two time ranges (From1 to To1 and From2 to To2) and supply the mean levels of the ranges (Mean1 and Mean2). The data in the time ranges must have significantly different levels and the mean values you set must not be the same.



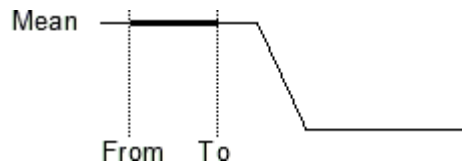
Values at two times

You define two times (At1 and At2) and the two values (Value1 and Value2) that correspond. The two values must be different and the data at the two times must also be different.



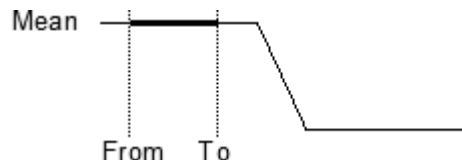
Set offset from mean of time range

If your data is calibrated, but suffers from baseline drift, you can use this method to redefine your base line. Set a time range (From and To) and the mean value of the data in the range (Mean). The data scaling is not changed.



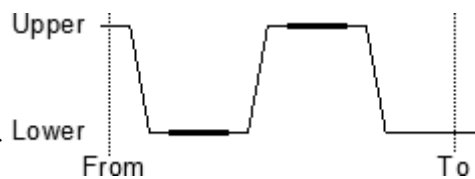
Set scale from mean of time range

This method is for data with a fixed offset (usually 0) and variable gain. Set a time range (From and To) and the mean value of the data in the range (Mean). The data zero level is preserved. The mean value you set cannot be 0 and the mean level of the data before calibration cannot be 0.



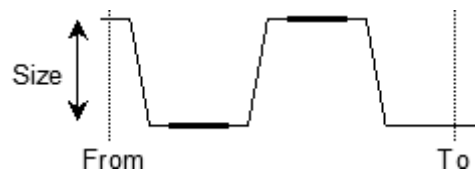
Square wave, upper and lower level

This calibrates a square wave with known Upper and Lower levels; Upper is the larger value before calibration. The time range (From and To) must contain at least three transitions. Spike2 detects the transition points between high and low values. For each high or low section, the first and last 25% of the data is ignored. The upper and lower levels must differ.



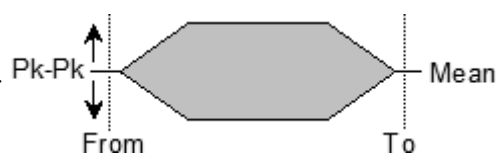
Square wave, amplitude (Size) only

This method detects a square wave in the same way as the Square wave upper and lower level method. In this case, you supply the Size (difference between the upper and lower levels) of the waveform. The zero level is preserved.



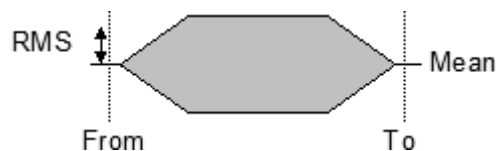
Peak to peak amplitude and mean

This method calibrates based on the peak to peak amplitude (Pk-Pk) and Mean value of the data in the time range (From and To). This could be used for a sinusoidal waveform of known amplitude. The Peak to Peak value must be non-zero.



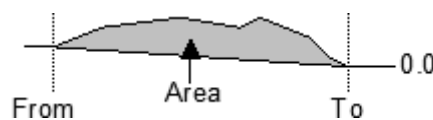
RMS amplitude about mean

This method calibrates based on the RMS (Root Mean Square) amplitude of the data in the time range (From and To) and the Mean level. The RMS value must be non-zero.



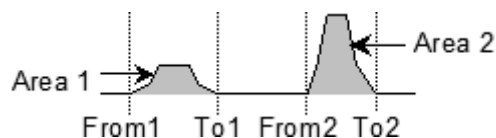
Area under curve, assume zero at end

Use this method to calibrate a rate based on a known area (for example flow rate based on volume). The rate is assumed zero at each end of the time range (From and To) to allow for a drifting base-line; the base line is assumed to run linearly from From to To. Set the area as rate units times seconds.



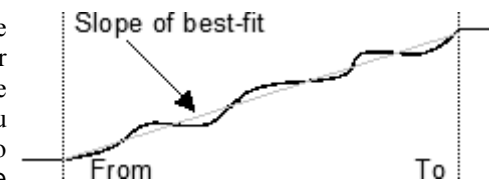
Areas under curve, two time ranges

Use this method when you know the area under the data for two sections (From 1 to To 1 and From 2 to To 2) of your data. The mean level of the two sections must be different. The area you set is in the units of the channel units times seconds.



Set scale from slope (no offset change)

Use this method when you know the slope of a section of the data (for example to calibrate position on a known velocity, or velocity on a known acceleration). The calibration sets the slope of the best-fit line to the data in the range to the slope value you set. This method does not change the offset, so you may need to combine this with the Set offset from mean of time range method for a full calibration. The units of the slope you set are the channel units per second.

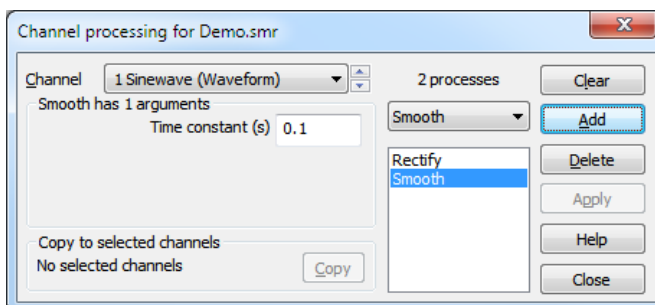


Channel process

A channel process is an operation, for example rectification, applied dynamically to waveform or RealWave data channels. The original data is not changed, but all users of the channel see data modified by the process. Multiple processes can be applied, for example DC removal then rectification then smoothing. Every time you use the data, the processing is applied, so using a processed channel is slower than using raw data. If a channel has an attached process, the channel number is displayed in red.

If the wave is already part of an analysis operation, such as a waveform average, adding a process and continuing the analysis may generate incorrect results. This is particularly the case with processes that change the sample rate or interpretation of the data. Restart the analysis operation to ensure correct results.

To add a process, open the Channel Process command from the Analysis menu and select a Channel or right-click the target channel and select Channel Process. The list to the left of the Apply button shows processes to apply in order. Editable values for the selected process are displayed on the left of the dialog. Clear removes all processes from the channel. Add appends a new process set by the drop down list to the left of the Add button to the end of the list. Delete removes the selected process. Apply copies changed argument values to the process.



Copy to selected channels

You can copy all the processes set for the current channel to other channels. Select the target channels by clicking their channel numbers (usually on the left edge of the time view) and click the Copy button. If the current channel has no processes, this will clear all processes from the selected channels.

Restrictions caused by processing

Some processes change the channel scale and offset (these are the values that translate between a 16-bit integer representation of a waveform and user units). Such changes do not affect the data on disk and are removed when the process is removed. The Calibrate dialog and the Channel Information dialog will not allow you to change the channel calibration if an attached process has changed the channel scale or offset. You are not allowed to calibrate a RealWave channel that has any attached channel process.

You can add processes of the following types: Rectify, Smooth, DC Remove, Slope, Time shift, Down sample, Interpolate, Match channel, RMS amplitude, Median filter, Fill gaps, No Nan

Rectify

This replaces all negative input values with positive values of the same magnitude. The result of this operation may exceed the 16-bit range of a waveform channel if the channel offset is negative, in which case the output will be limited to the available 16-bit range. There are no additional arguments required to define this process. This is a non-linear process, in the sense that the output is not related to the input by a transform of the form $output = scale * input + offset$.

Smooth

This process has one argument, a time period in seconds, p . The output at time t is the average value of the input data points from time $t-p$ to $t+p$ seconds. This process does not affect the channel scale or offset. More about smoothing.

DC Remove

This process has one argument, a time period in seconds, p . The output at time t is the input value at time t minus the average value of the input data points from time $t-p$ to $t+p$, that is, it is equivalent to the original signal minus the result of the smoothing operation, described above. This process does not affect the channel scale, but the channel offset is set to zero.

Slope

This process has one argument, a time period in seconds, p . The slope at time t is calculated using an equal weighting of the points from time $t-p$ to $t+p$. This is done by calculating the mean of the points ahead and the mean of the points behind each data point, and the slope is taken from the line through the centre of the points behind to the centre of the points ahead. This calculation is equivalent to applying an FIR filter, it is not a least squares fit through the points. This method is used because it can be applied iteratively very efficiently to a long run of data. If you apply this process to a channel, the channel scale, offset and units change. If the current channels units are no more than 3 characters long, "/s" is added to them, so units of "V" become units of "V/s". If there is not sufficient space, the final character of the units becomes "!" to indicate that the units

are no longer correct. The offset becomes 0, and the scale changes to generate the correct units.

Time shift

This process has one argument, the time to shift the wave. A positive time shifts the wave into the future (to the right); a negative time shifts the wave into the past. If you want to keep the waveform unshifted, but change the positions where the data was sampled, use the **Interpolate** or **Match channel** processes.

Down sample

This process changes the sample rate of the wave by taking one point in n . There is one argument, prompted by **Use one point in**, which is the down sample ratio. You might want to use this command after filtering or smoothing a waveform. This is a faster operation than **Interpolate**.

Interpolate

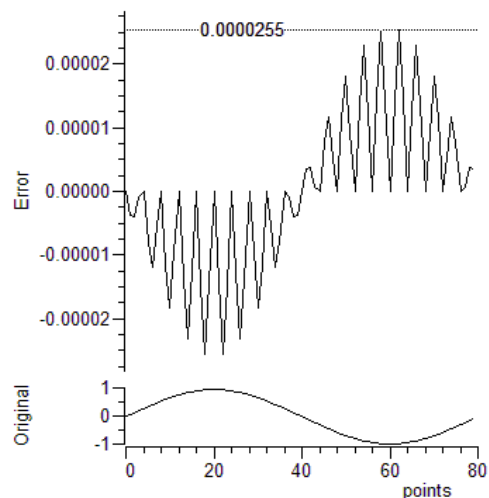
You can change the sample rate of a channel and set the time of the first data point with this process. Interpolation is by cubic splining the original data. No data is generated outside the time range of the original data points. Interpolation is not too slow, but if you increase the sampling rate it will take longer to draw and process data.

The first argument, **Sample interval**, is the time in seconds between output data points. You can type in expressions here so, to set 123 Hz, type $1/123$. The actual interval is set as close to the requested one as possible. When you create the process this is set to the current sample interval of the channel.

The second argument, **Align to**, aligns the output data to a time. It must be positive. The process places data points at this time plus and minus multiples of the sample interval. You can use this to convert multi-channel data sampled by a 1401 into simultaneously sampled data by giving all the channels the same alignment and sample rate.

Cubic spline interpolation assumes that the input waveform and its first and second differentials are continuous. If the input data was suitably filtered this will be not too far from the truth. Not all data is suitable for cubic splining; splining across step changes generates ringing effects that were not present in the original signal.

Cubic spline interpolation is better than linear interpolation for continuous data, but it is not perfect. The graph shows the error between a sine wave sampled with 20 points per cycle and splined to 80 points per cycle and a calculated 80 points per cycle sine wave. The maximum error is small, in this case 0.0000255 of the original signal. However, the maximum error increases rapidly the fewer points there are per cycle of the original data. With 5 points per cycle, the maximum interpolation error for a sinusoid is almost 1 per cent of the original signal.



Match channel

This is the same as **Interpolate**, except that the sample interval and alignment are copied from a nominated channel. The initial channel is set to the current channel, so adding this process should have no visible effect (apart from causing a redraw).

RMS amplitude

This process has one argument, a time period in seconds, p . The output at time t is the RMS value of the input data points from time $t-p$ to $t+p$ seconds. For waveform data, the output may be limited by the 16-bit nature of the data if the channel offset is non-zero.

Median filter

This process has one argument, a time period in seconds, p . The output at time t is the median value of the input data points from time $t-p$ to $t+p$ seconds. The median is the middle point after the data has been sorted into order. This can be useful if your data has occasional points with large errors. This filter is slow if p spans a large number of data points; set the time period to the smallest value that removes the outlier points.

Fill gaps

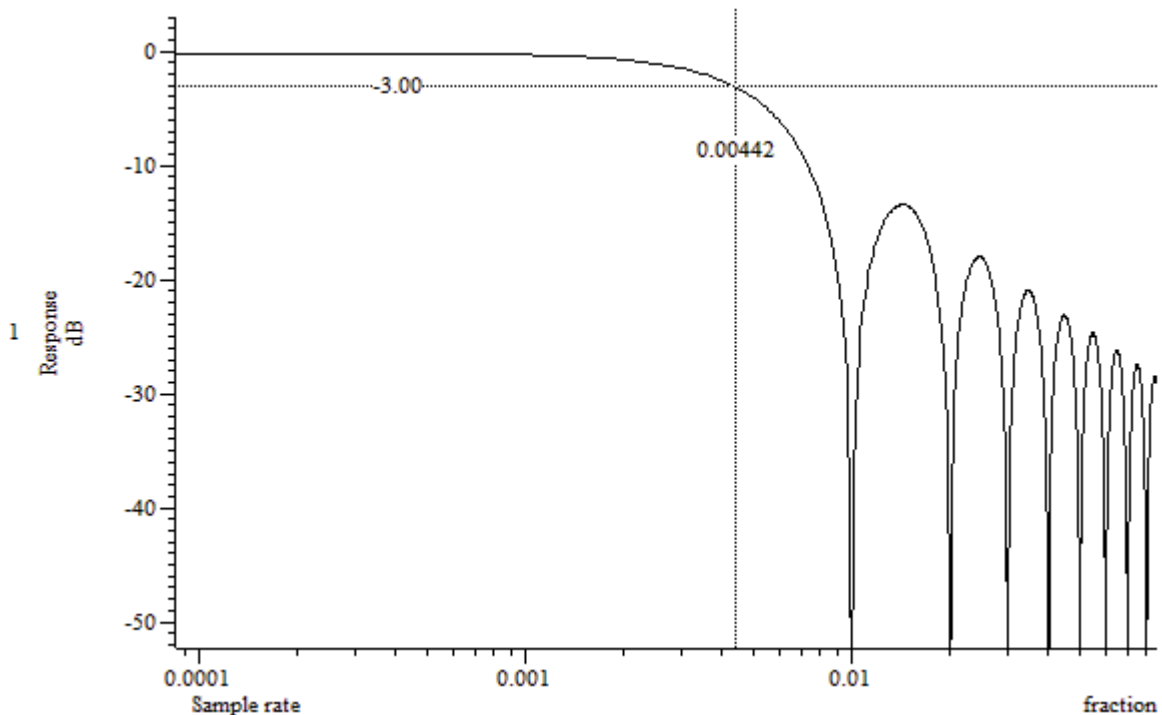
Waveform and RealWave channels can have gaps, where no data exists. This process guarantees that the channel has no gaps by filling in gaps greater than a specified time with a fixed level and by linearly interpolating across smaller gaps. There are two arguments: the maximum gap in a waveform to interpolate across and the level to use to fill gaps wider than the maximum gap. At the start and end, gaps less than the specified maximum are filled by duplicating the first or last data point. This is a non-linear process (see Rectify, above).

Skip NaN

A NaN (Not a Number) is a floating point value that is either undefined in some way, or is infinite (the result of dividing by zero). This can occur when RealWave data is imported into Spike2. This process removes such numbers from the channel, leaving gaps. Applying this process to a Waveform channel is a waste of processing time. The process has no arguments.

More about smoothing

The smoothing function is equivalent to an FIR filter with n coefficients, each of value $1/n$. This generates a filter with the following characteristics:



*Frequency response of smoothing filter ($n=100$).
Frequency expressed as a fraction of the sampling rate.*

This is not a particularly good filter, but it suffices for many purposes and has the advantage that it can be realised efficiently. The position of the zeros is determined by the number of coefficients and by the sampling rate. If the sampling rate is s and there are n coefficients, the zeros occur at $i*s/n$ Hz, where i is 1, 2, 3... If we consider the first section of the filter, the 3 dB point is at approximately 0.44 of the position of the first zero. If we express this in terms of the smoothing dialog, if you enter a time period t , the first zero occurs at a frequency given by $1/(2*t+si)$ where si is the sample interval of the channel. If si is small compared to t then this is $0.5/t$ Hz. The 3 dB point occurs at $0.44/(2*t+si)$ or $0.22/t$ Hz if si is small compared to t .

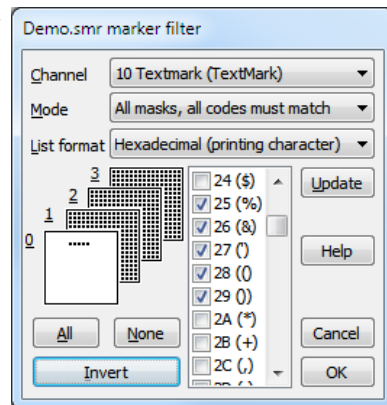
The graph was made with the following script. You can adjust the frequency response by changing the `np%` parameter. The frequency axis is shown as a fraction of the sampling rate.

```
const np% := 100;      'Data points used for smoothing
const bins% := 4097;  'size of the response array
var coef[np%];        'array to use to simulate the smooth filter
arrconst(coef, 1.0/np%); 'Set coefficients all the same
var rv%;
```

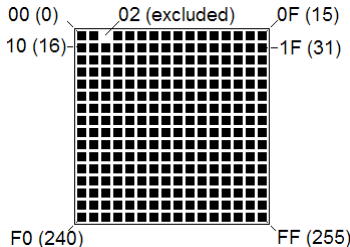
```
rv% := SetResult(bins%, 0.5/(bins%-1), 0, "Frequency response",
    "fraction", "dB", "Sample rate", "Response");
FIRResponse([], coef); 'calculate the response
DrawMode(1,2); 'draw as line
YRange(1,-40,0); 'set y range in dB
XAxisAttrib(1); 'Log the x axis
XRange(0.0001, 0.5); 'set a sensible range
WindowVisible(1); 'make it visible
```

Marker Filter

You can filter any channel that holds marker, WaveMark, TextMark or RealMark data. Each of these channels has a *marker filter* that selects the data items to display and use in calculations. Each marker data item has four marker codes that are matched against the marker filter. If you set a marker filter so that it would not pass all data, the channel number is displayed in red.



Masks



The dialog shows the marker filter as four masks numbered 0 to 3. Each mask has 256 elements in a 16x16 grid, one for each possible code value. The contents of the front-most mask are displayed in the scrolling list in the centre of the dialog. Click on a mask element to bring the mask to the front and scroll the list to the element.

The top row of each mask represents code values 0 to 15 (hexadecimal codes 00 to 0F), the second row 16 to 31 (10 to 1F) and so on down to the bottom row, which represents values 240 to 255 (F0 to FF). If a code value is included in the filter, the corresponding element is black. When values are excluded, the element is white. The mask gives a quick indication of the state of the filter. There are three buttons that act on the entire mask:

- All This includes all code values in the filter (checks all the list boxes)
- None This excludes all code values from the filter (clears all the list check boxes)
- Invert This excludes included values and includes excluded values

Channel

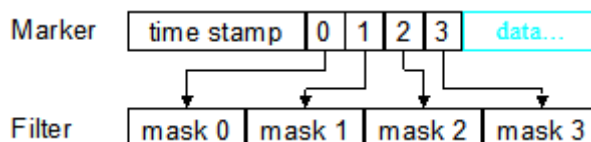
This field is the standard channel selector. You can select only marker-based channels.

List format

The list shows each the code value as two hexadecimal or one to three decimal digits, the single character equivalent or as a combination as set by the *List format* field. Characters are appropriate for keyboard markers, numeric digits are used for the non-printing codes or can be forced for all codes. If you type a character, the list scrolls to the entry that starts with the typed character. To include marker codes, check the boxes. There are two modes in which to use the marker filter:

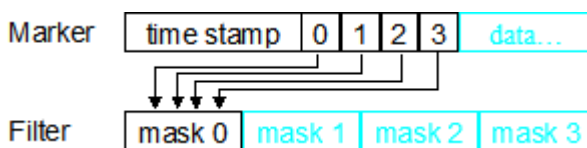
Mode 0 (AND): All masks, all codes must match

For a data item with marker codes *a*, *b*, *c* and *d* to be included, mask 0 must have code *a* checked, mask 1 must have code *b* checked, mask 2 must have code *c* checked and mask 3 must have code *d* checked. Most users of this mode set mask layers 1, 2 and 3 to All and use the first layer to select data values. You can think of this as the AND mode; to accept data marker code 0 must be in the mask 0 AND marker code 1 must be in mask 1 AND marker code 2 must be in mask 2 AND marker code 3 must be in mask 3.



Mode 1 (OR): One mask, any code can match

Only mask 0 is used, the rest are greyed out. For a data item with marker codes *a*, *b*, *c* and *d* to be included, mask 0 must have one or more of the codes *a*, *b*, *c* and *d* checked. You can think of this as the OR mode; to accept data marker code 0 OR marker code 1 OR marker code 2 OR marker code 3 must be in mask 0. There is one exception; for marker codes 2 to 4, the code 00 is ignored. To accept code 00, it must be the first marker code.



Mode 1 is often used when sorting spike shapes (WaveMark data) and you discover a WaveMark that is the result of a collision between two spikes. You can set the first marker code to the code for the first spike and the second to the code for the second (leaving the third and fourth codes as 00), then the spike will appear on screen and in analyses when either of the codes are included in the mask.

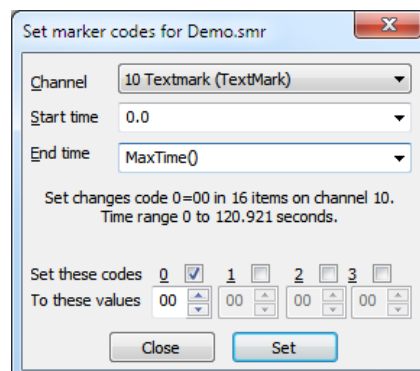
Cancel, Update, Help, OK

The buttons on the right are:

- Update Update the channel display to correspond with the new filter
- Help Display this information
- Cancel Close dialog, cancelling changes since last update
- OK Close dialog, accept new filter

Set Marker Codes

You can open this dialog from the Analysis menu, by right clicking on a marker channel and from the Edit WaveMark dialog. From here you set selected marker codes for marker, WaveMark, TextMark and RealMark channels. Each data item has four marker codes that can be used to filter the data in the channel. Usually only the first marker code is used, see the Marker Filter command for more information. The Channel field sets the channel to process and the Start time and End time fields identify the markers to be set. If a marker filter is set for the channel, only data items that are in the filter are changed by this command. The four check boxes 1 to 4 select the codes to change. You set the value for each code as either two hexadecimal digits or one printing character.



Example

To change all the marker items in a channel with a first marker code of 02 to have a first marker code of 1A do the following:

1. Open the marker filter dialog and set the channel to display only marker code 02.
2. Open this dialog, select the channel and set the time range as 0.0 to Maxtime().
3. Check the 1 box to set the first marker code and edit the text under the check box to 1A. Make sure the other check boxes are clear.
4. Read the text in the centre of the dialog as a check that this is the action you wish to take. This action cannot be undone.
5. Click the Set button to change the marker codes.

All the code 02 items will disappear as they have been coded as 1A, so you will need to change the marker filter settings if you want to see the result.

Selecting WaveMark data with the mouse

Whenever WaveMark data is displayed as a waveform, you can select one or more events using the mouse and then change the marker codes with a dialog very similar to the one displayed above except that the Channel, Start time and End time fields are omitted as the selection of events is done by line crossings. To select the events, hold down **Ctrl+Alt**, then click and drag a line to intersect the WaveMark items that you want to select.

New WaveMark

This creates WaveMark data channels from waveform data or existing WaveMark data. You can also activate this option by right-clicking on a waveform or WaveMark channel and selecting the New WaveMark command from the context menu.

New Stereotrode, Tetrode

Use this to create a new WaveMark channel with 2 or 4 traces. To enable this option, select 2 or 4 waveform channels with the same sample rate. This option opens the New WaveMark dialog with the selected channels as the source (see Spike sorting for details). The channels are used in the order that they are selected. The first selected channel sets the sample rate that the selected channels must match.

If the selected channels have the same units but different scales, we will scale the data to match the channel with the largest scale factor. If the difference in scales between the channels exceeds a factor of 20 you are warned and offered the chance to abandon the operation. You can edit channel scaling in the Channel Information dialog.

If the selected channels have different units, we display a warning dialog. If you choose to continue we pretend that all the channels have the same scales and units, being the values for the first channel in the list. You can edit the units for a channel in the Channel Information dialog.

Edit WaveMark

This Analysis menu option reclassifies WaveMark data both manually and automatically. You can also activate this option by right-clicking on a WaveMark channel and selecting the Edit WaveMark command from the context menu.

Digital filters

This displays a pop-up menu in which you can choose the FIR Digital filtering dialog or the IIR Digital filtering dialog. From these dialogs you can create digital filters and apply them to waveform or RealWave channels. You can also open these dialogs by right clicking on a suitable channel and selecting a digital filter option from the context menu.

10: Window menu

Window menu

The Window menu has five permanently present commands, one to duplicate a time window and the remainder to arrange the windows. The remaining space in the menu holds a list of all the windows that belong to the Spike2 application. If you select one of the windows in the list, the window is brought to the front and made the current window.

Duplicate window

This command is only available within a time window and creates a duplicate window with all the attributes (list of displayed channels, event display modes, colours, cursors and size) of the original window. Once you have created the new window, it is independent of the original. However, the data channels within it are the same data channels as in the original window, so any changes made to the data in one window will cause all duplicated windows to update. Duplicating a window allows you to have different views of the same data file with different scales and drawing modes and different sets of channels.

You can close all windows associated with a data document using the control key plus close. This will remember the position and state of all windows associated with the document.

Hide

This command makes a window invisible. This is often used with script windows and sometimes is used to hide data windows during sampling when only the result views are required. Closing the Log window is equivalent to hiding it as the Log window always exists.

Show

This option lists all hidden windows in a pop-up menu. Select a window from the list to make it visible.

Tile Horizontally

This command arranges all the visible screen windows so that no window is overlapped by any other. Iconised windows are arranged along the bottom edge of the window. The command attempts to arrange window so that they are wider than they are high. Tiling takes into account the space used for title bars of iconised windows, so to use the full application window area you should hide any iconised windows first. This command may have the same result as **Tile Vertically**, depending on the number of windows.

Tile Vertically

This command arranges all the visible screen windows so that no window is overlapped by any other. Iconised windows are arranged along the bottom edge of the window. The command attempts to arrange windows so that they are taller than they are wide. Tiling takes into account the space used for title bars of iconised windows, so to use the full application window area you should hide any iconised windows first.

Cascade

All windows are set to a standard size and are overlaid with their title bars visible. Any iconised windows are left in the iconised state, and they are arranged along the bottom edge of the window, as for the **Arrange Icons** option.

Arrange Icons

You can use this Window menu command to tidy up the windows that you have iconised in Spike2. The icons are lined up along the bottom edge of the application window.

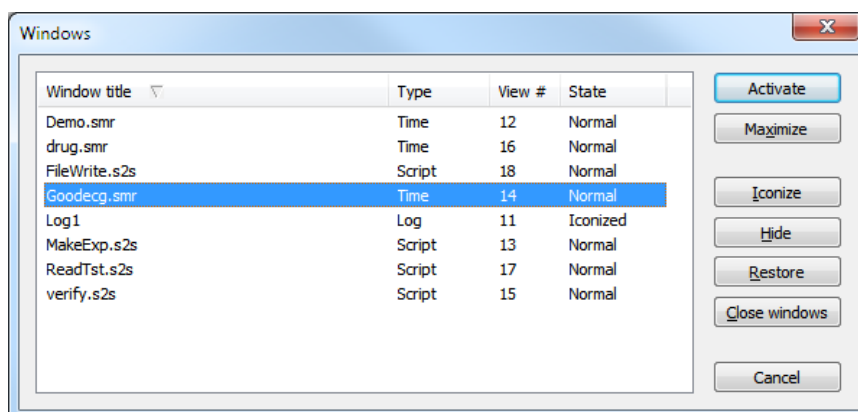
Close All, Close All and Link

This command closes all windows in the Spike2 application. You are asked if you want to save the contents of any text windows that have changed or any newly sampled data window. You can avoid being asked if you want to save modified result and XY windows with an option in the Edit menu Preferences.

If you hold down the Ctrl key before clicking on the Menu, this option becomes **Close All and Link**. In this case, time views will also save sufficient information about attached result and XY views to restore them when the time view is opened. This extra information is saved in the resource file associated with the time view (and this can be a lot of information). This is equivalent to the File menu **Close and Link** command applied to all open files.

Windows

This dialog lists all the document-related windows that are open and lets you apply common window operations to one or more of the windows. You can sort the list based on the window title, type, view number (as seen by the script language) and window state by clicking the title bar at the top of the list.



11: Cursor menu


Cursor menu

The cursor menu creates and destroys vertical and horizontal cursors, changes the display to make them visible, changes their labelling mode and obtains the values of channels where they cross vertical cursors and between vertical cursors. Up to 10 vertical (numbered 0-9) and 9 horizontal (numbered 1-9) cursors can be active in each time or result window. Cursors in separate windows are independent of each other. When a window is duplicated, the cursors are also duplicated, but are then independent of the original view.

Vertical cursors

The following commands in the Cursor menu are used to control and interact with the vertical cursors. There are also vertical cursor context commands available by right-clicking on a vertical cursor. The vertical cursor commands available from the menu are:

New Cursor

This command and the  cursor button at the bottom left of time and result views adds a vertical cursor with the current label style and lowest available cursor number (1 to 9) to the window. The keyboard short cut `Ctrl+n` where `n` is a cursor number (including 0 in a time window) creates cursor `n` if it doesn't exist and moves it to the centre of the window. Each view has a default label mode for new cursors set by the `Label mode` command.

Delete

The delete command activates a pop-up menu from which you can delete one or all vertical cursors. As an aid to identification cursors are listed with their number and position. Deleting a cursor removes it from the window; other cursors are not affected. Deleting cursor 0 just hides it as cursor 0 always exists. To restore cursor 0 use the `Ctrl+0` key combination or the cursor menu `Fetch` command.

Fetch

This opens a pop-up menu in which you select a vertical cursor to place in the visible x axis. The keyboard short cut `Ctrl+n` where `n` is a cursor number will fetch cursor `n` if it exists or will create it if it doesn't exist. You can use this command to make cursor 0 visible if it is hidden.

Move To

This command opens a pop-up menu from which you can select a vertical cursor to move to. The cursors are listed with their number and position as an aid to identification. The window is scrolled to display the nominated cursor in the centre of the screen, or as close to the centre as possible. This command does not change the x axis scaling. You can also use the keyboard short cut `Ctrl+Shift+n` where `n` is a cursor number.

If cursor 0 is hidden you can still move to it. If you want to make cursor 0 visible use the `Fetch` command or the `Ctrl+0` key combination.

Position Cursor

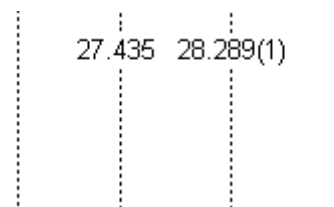
This command opens a pop-up menu to select a cursor and then opens a dialog in which you can type or select a cursor position. You can also activate this dialog by right clicking on a vertical cursor and selecting `Set Position` from the cursor context menu.

Display all

This command has no effect if there are no cursors. With a single cursor, the command behaves as though you had used the **Move To** command to select it. With multiple cursors, the window is scrolled and scaled such that the earliest cursor is at the left-hand edge of the window and the latest is at the right-hand edge.

Label mode

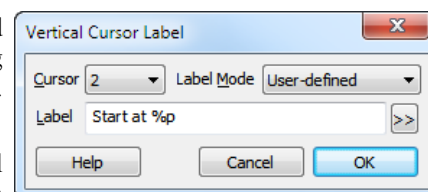
Each cursor has an optional label used to identify it. You can drag the cursor labels up and down the cursor with the mouse to suit the data. There are five cursor label modes: **None**, **Position**, **Number**, **Position and Number**, and **User-defined**. You select the most appropriate for your application using the pop-up menu or by right clicking on a cursor and choosing to set the cursor label from the context menu. To avoid confusion between the cursor number and the position, the number is displayed in bold type when it appears alone and bracketed with the position.



Each cursor stores its own mode and label, and the view has a mode that is applied to new cursors. The first four items in the menu set the view mode and the mode of all cursors. You can also set a user-defined label for cursors (but not for the view) with the **Set Label** command.

Set Label

You can open the cursor label dialog from the **Label Mode** and **Horizontal Label Mode** cursor menu commands or by right-clicking on a cursor and using the **Set Label** command in the context menu. The dialog for horizontal cursors differs only in the dialog title.



From this dialog you can set the cursor label mode for one or all cursors. The **Cursor** field can be set to the number of any cursor in the view, or **All**. If you choose **All**, any change applies to all cursors and sets the view mode except in **User-defined** mode, where the view mode does not change.

The **Label Mode** field lets you choose one of **None**, **Position**, **Number**, **Position and Number**, and **User-defined** as the cursor mode. If you select **User-defined**, the **Label** field appears together with the **>>** button and you can set a label of your choosing.

User-defined labels

User-defined labels display the text you type, except that text sequences introduced by **%** are replaced:

- %p** Replaced by the cursor position
- %u** Replaced by the position units as displayed on the axis
- %q** Equivalent to **%p %u** (position followed by units)
- %n** Replaced by the cursor number
- %v(n)** Replaced by the channel *n* value at the cursor. Not valid for horizontal cursors or in an XY view.
- %w(n)** The same as **%v(n)** but followed by the units of the value.

You can also stipulate the width (**w**) and the number of decimal places (**D**) used for the position and value by using **%W.Dp** and **%W.Dv(n)**. For example: **%n at %6.4p %u, %v(2)** might display: **1 at 2.2346 s, 87.128756** if cursor 1 was at 2.2346 seconds and channel 2 had the value 87.128756 at this point. The **%v(n)** option is not allowed for horizontal cursors or for vertical cursors in an XY view. The value returned by **%v(n)** is the same value as displayed on the axis for that cursor and channel.

The **>>** button pops up the list of replacements, and if you choose one, it replaces any selection in the **Label** field. If you choose the **%v(n)** option, you are prompted to select a channel to measure.

We allow you to type in quite long labels. Before Spike2 7.11 labels longer than 19 characters were truncated when you closed a data file. However, very long labels look messy; it is usually better to keep labels short.

User-defined labels and drop down lists

If a cursor has a user-defined label, it is used as a tool tip when you select the cursor from a drop-down list. To try this, go to the Active cursor dialog. Set several vertical cursors with user-defined labels, select a search mode, then open one of the drop-down lists of x axis positions and hover the mouse over one of the `Cursor(n)` items in the list. If the cursor has a user-defined label, the label will appear as a tool tip.

Scaling of positions and units

If the associated axis is in an Auto units mode, the displayed label and units match the axis. This is a purely cosmetic effect and does not change the value used in the `Cursor()` or `HCursor()` script command.

Renumber

When created, cursors take the lowest available cursor number. This command renumbers the vertical cursors 1 through 9 so that the lowest numbered cursor is on the left and the highest on the right. Renumbering cursors has no effect on cursor 0. As currently implemented, all other cursor features (labels, active cursor mode) are unchanged by renumber, only the cursor number changes. Because of this, be cautious about using the renumber command with active cursors.

Active cursors

In a time view, vertical cursors can be *active* or *static*. An active cursor can seek to a position based on the position of other cursors and data in a channel. Active cursors can automate data analysis, leading to new data channels, XY trend plots or tabulated output.

Cursor 0

Vertical cursor 0 is special. It always exists and cannot be deleted, but it can be hidden. It is the iterator for XY view and data channel measurements; a movement of cursor 0 causes all other active cursors to recalculate their positions. This calculation is done in order of rising cursor number. Cursor 1-9 recalculate their position if cursor 0 moves. To hide cursor 0, right click on it and select **Hide cursor 0** in the context menu. Script users can cause a nominated range of cursors to search with the `CursorSearch()` command.

Valid and invalid cursors

Active cursor positions are either *valid* or *invalid*; invalid cursors have an exclamation mark at the end of the label. A position is invalid if a search fails and the **Position if search fails** field is empty or does not contain a valid expression. Expressions that use invalid cursor positions are also invalid. The measurement system rejects invalid positions. Cursor positions are validated by operations that move them to a specific place such as dragging. If a search results in an invalid position, the cursor is not moved. Script users can test if a cursor has a valid position with the `CursorValid()` command.

Cursor positions after a failed search

Cursor 0 does not move after a failed search. The remaining cursors are set to the mid-point of the search region if a search fails. We may change this position in the future, or give you more control over it, so please do not rely on the position of an invalid cursor.

Active mode

This command opens the Cursor mode dialog (script equivalent `CursorActive()`). The Search method field and the selected cursor determine the dialog contents. An active cursor has an associated Search channel and start and end positions that define the data to search to locate the new cursor position.

Cursor 0 does not have start and end positions. However, it does have a Minimum step; searches start at this distance from the cursor 0 position and continue to the file end for a forward search, or to the file start file for a backward search. Cursor 0 has a restricted range of search methods: Peak find, Trough find, Rising threshold, Falling threshold, Within thresholds, Outside thresholds, Peak slope, Trough slope, Turning point, Data points and Expression.

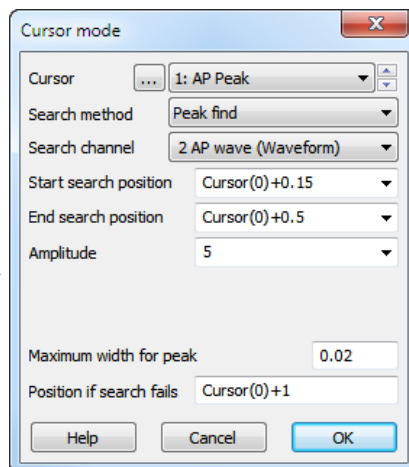
The search positions can be a fixed time, but more usually they will be expressions that involve the positions of other active cursors. Active cursor positions are evaluated in sequence from cursor 0 to cursor 9.

For cursor n , an expression that refers to an active cursor less than n uses the new position. An expression that refers to a cursor greater than or equal to n uses the old position.

If the End position for search is less than the Start position for search, searches are backwards. If a search is backwards, read *previous* for *next* and *last* for *first* in the descriptions of the cursor modes. Where search modes depend on data values, for example maximum and minimum, and the data channel does not have a y axis, then the value is taken as the interval between data items.

Several modes use a slope. These modes have the value Width for slope measurement, in seconds and use the data points from $Width/2$ before the current position to $Width/2$ after to calculate the slope unless there are more than 200 points, in which case 100 points before and after are used. The contribution of each point to the slope is proportional to the distance of the point from the current position. Because the slope at any point requires data around it, the slope within $Width/2$ of the ends of the data and any gaps is not to be relied on as the missing data is assumed to be zero.

The ... button next to the cursor number opens the Cursor Label dialog where you can set a user-defined label (sometime useful to make it clear which cursor is in use).

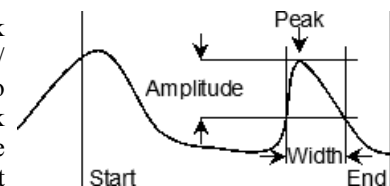


Static

A new cursor starts out in Static mode. It stays where you put it and is not changed by a change in the position of a lower numbered cursor. The cursor position is always valid.

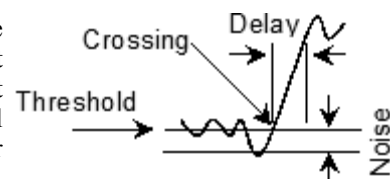
Peak find, Trough find

The Amplitude field defines how much the data must rise before a peak and fall after it (or fall before a trough and rise after it), to detect a peak/trough. The Maximum width for peak field rejects peaks that are too broad (set it 0 for no width restriction). For waveform channels, the peak position is located by fitting a parabola through the highest point and the points on either side. In the diagram, which shows a peak search, the first peak is not detected because the data did not rise by Amplitude within the time range. The cursor position is valid if a peak is detected, invalid if not.



Rising threshold, Falling threshold, Threshold

The data must cross Threshold from a level that is more than Noise rejection (hysteresis) away from it and stay crossed for a time of at least Delay after level crossing. For the Rising threshold mode, the data must increase through the threshold, for Falling threshold mode it must fall through the threshold. In Threshold mode the crossing can be in either direction. The picture shows a rising threshold. For waveform channels, the crossing point is found by linear interpolation of the data points on either side of the threshold crossing. The cursor position is invalid if a crossing does not occur.



Outside thresholds, Within thresholds

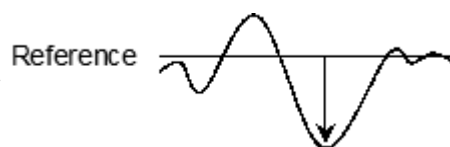
These two modes are similar to the rising and falling thresholds modes except they search for data that lies outside two threshold levels or within two threshold levels. In addition to the Threshold level and Noise rejection (hysteresis) fields, there is a Threshold level 2, which sets the second threshold level. The data must cross a threshold from a level that is more than Noise rejection away from it for a period of at least Delay.

Maximum value, Minimum value

The result is the position of the maximum or minimum value. It is invalid if there is no data in the search range.

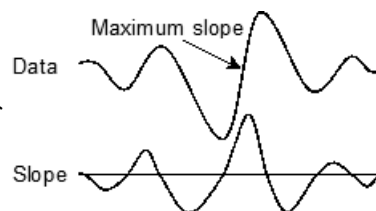
Maximum excursion

There is an extra field in this mode for the Reference level. The cursor is positioned at the point that is the maximum distance in the y direction away from the reference level. The result is invalid if there is no data in the search range.



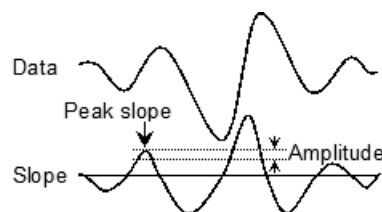
Steepest rising, Steepest falling, Steepest slope (+/-)

These modes are for waveform channels only. They have the extra field Width for slope measurement that sets length of data used to evaluate the slope at each data point. The result is the position of the maximum, minimum or maximum absolute value of the slope. The result is invalid if there is no data or not enough data to calculate a slope or if the channel is not waveform or WaveMark.



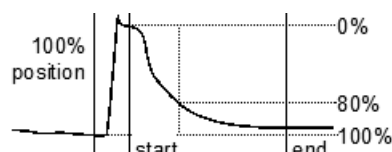
Slope peak, Slope trough

These analysis modes find the first peak or trough in the slope within the search range that meets the Amplitude specification. The Width for slope measurement field sets the length of data used to evaluate the slope at each data point. The Amplitude field sets how much the slope must rise before a peak and fall after it (or fall before a trough and rise after it), for it to be accepted as a peak. The Amplitude units are y axis units per second.



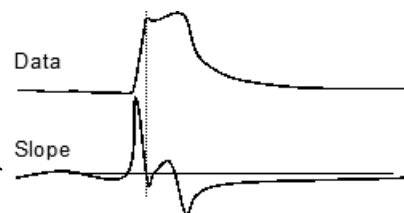
Repolarisation %

This mode finds the point at which a waveform returns a given percentage of the distance to a baseline inside the search range. The start of the search range defines the position of 0% repolarisation. The 100% position, which can be anywhere, and Width fields identify the 100% level. The Repolarise % field (drawn at 80% in the picture) sets a threshold level in percent relative to the 0% and 100% levels. The position is the first point in the search range that crosses the threshold. The result is invalid if the threshold is not crossed.



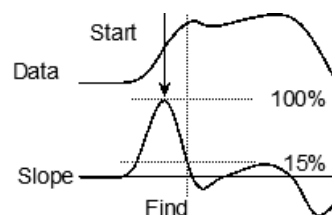
Turning point

This finds the first point in the search range where the waveform slope changes sign. Put another way, it finds a localised peak or trough. The Width for slope measurement field sets the data range to calculate the slope. The picture shows how this can find the top of a sharp rise where Maximum would get the wrong place. To use this, set a cursor on the peak slope and start the search from that point. The result is invalid if no point is found.



Slope%

Use this to find the start and end of a fast up or down stroke in a waveform. The Width for slope measurement field sets the time width used to calculate the slope. The Slope% field sets the percentage of the slope at the start of the search area to find. Set a cursor on the maximum or peak slope, then use that as the start point and search for the required percentage. A value of 15% usually works reasonably well. The result is valid if the slope value is located.



Data points

The Point count field sets the number of data points to move by in the search range on the referenced channel. The result is the time of this data point. The cursor position is invalid if there is no data item in the search range.

Expression

The new position is obtained by evaluating the Position field, which normally holds an expression based on cursor positions, for example $\text{Cursor}(0)+1$ or $(C0+C1)/2$.

Search Right, Search Left

If cursor 0 is active, these two commands cause the cursor to search for the next or previous position that satisfies the active mode. If the cursor mode is Expression, the cursor goes the same way for both commands.

Cursor values



This command opens a new window containing the values at any cursors in the current time or result window. Cells for cursors that are absent, or for which there is no data, are blank. The new window is a top level window and will never go behind another time or result view.

Cursors	Cursor 0	Cursor 1	Cursor 2	Cursor 3
Time (s)	36.2763	42.32235	48.3684	54.41445
31 Keyboard	37.04064	43.2512	48.48128	56.33024
3 Response	36.36736	42.34483	48.55471	54.56006
2 Stimulus	36.29368	42.36524	48.47603	54.4575
1 Sinewave	1.9878232	1.4949183	1.382745	1.4771444
<input type="checkbox"/> Time Zero	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input type="checkbox"/> Y Zero	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

In a result window, the value displayed is the value to be found in the bin to the right of the cursor. If you position the cursor at the far right (where there is no bin to the right), a blank is displayed as the value.

In a time window, the values displayed depend on the channel display mode. If a channel has a y axis the displayed value will be in y axis units. If the data is displayed as a continuous line or as a histogram, the displayed value is where the line crosses the cursor. If the data is displayed as dots, the value is the nearest dot for waveforms and the previous dot for instantaneous frequency. Positions more than one waveform sample interval away from any waveform point have no value.

If the channel has no y axis, in State drawing mode, the value is the state. Otherwise the channel holds an event or marker and the displayed value is the time of the next event or marker at or to the right of the cursor. If there is no data, the field is blank.

The Time zero check box enables relative cursor times. If checked, the cursor marked with the radio button is taken as the reference time, and the remaining cursor times are given relative to it. The reference cursor displays the absolute time, not 0. In the example above, cursor 1 has been set as the reference.

The Y zero check box enables relative cursor values. The radio buttons to the right of the check box select the reference cursor. The remaining channels display the difference between the values at the cursor and the values at the reference. The values for the reference cursor are not changed.

Selecting copying and printing data

You can select areas of this window and copy them to the clipboard by clicking on them with the mouse. Hold down the Shift key for extended selections. You can select entire rows and columns by clicking in the cursor

and channel title fields. Hold down the Control key to select non-contiguous rows and columns. Use `Ctrl+C` to copy selected data to the clipboard. Use `Ctrl+L` to copy selected data to the log view. Right click in the cursor window to display a context menu with command for Copy, Log, Font and Print.

Cursor regions



This command opens a cursor region window for the current time or result view. This window calculates values for data regions between the cursors. One column can be designated the Zero region by checking the box and selecting the column with a radio button. The value in this column is then subtracted from the values in the other columns (except in Area(scaled) mode which scales the zero region value to allow for column widths).

Cursors	0 - 1	1 - 2	2 - 3
Time (s)	6.04605	6.04605	6.04605
1/Time (Hz)	0.16539724	0.16539724	0.16539724
31 Keyboard	0.33079449	0.16539724	0.33079449
3 Response	11.908601	4.79652	3.3079449
2 Stimulus	16.870519	11.247013	7.9390677
1 Sinewave	1.7852456	1.414731	1.3965115

Zero Region
 Mean

The field at the bottom left sets the measurement method; click it for a full list. Cells with no cursors or data are blank. You can interrupt long calculations in time windows with the `Ctrl+Break` key combination.

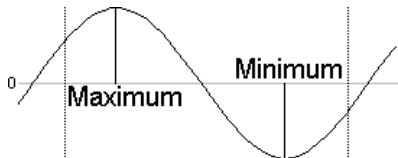
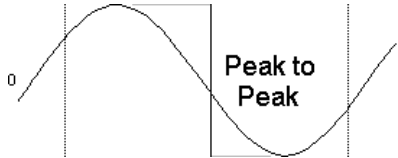
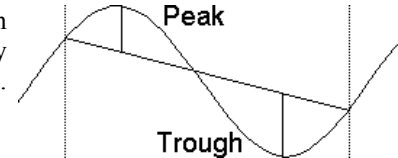
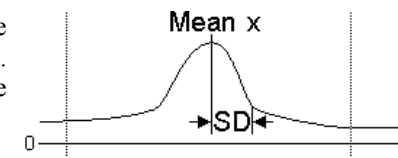
The effect of the selected measurement type depend on the view type and the channel type (time view waveform and event channels, result view channels) and, in some cases, the channel display mode.

The first time you open this dialog in a Spike2 session, it will be in Mean mode. Subsequent use of this dialog will open in the last mode that you set, as this is more likely to be the mode you want.

Waveform channels and result window data

In a time window, waveform channels include WaveMark channels drawn as waveforms, WaveMark or Cubic Spline and waveform and RealWave channels drawn as dots, Cubic spline or Sonogram. The region set by a pair of cursors is the data starting at the first cursor up to, but not including, the data at the second cursor. In a result window, the region set by a pair of cursors starts at the bin containing the left cursor and ends in the bin to the left of the bin containing the right cursor. The values for each mode are:

- Area** The area between the data points and the y axis zero. Area is positive for curve sections above zero and negative for sections below zero. Use **Modulus** if you want areas below the y axis to be treated as positive.
- Mean** The sum of all the data points between the cursors divided by the number of data points between the cursors. There is also a **Mean Absolute** value.
- Slope** The slope of the least squares best fit line to the data between the cursors.
- Sum** The sum of the data values between the cursors. If there are no samples between the cursors the field is blank.
- Area (scaled)** The same as **Area**, but if a zero region is specified, the amount subtracted from the other regions is scaled by the relative width of the regions.
- Curve area** Each data point makes a contribution to the area of its amplitude above a line joining the end points multiplied by the x axis distance between the data points. The picture makes this clearer.
- Modulus** Each data point makes a contribution to the area of its absolute amplitude value multiplied by the x axis difference between data points. This is equivalent to rectifying the data, then measuring the area. If a zero region is specified, the amount subtracted from the other regions is scaled by the relative width of the regions.

Maximum Minimum	For the maximum measurement, the value is the maximum value found in the time range between the cursors. For the minimum, the value is the minimum value in the time range.	
Abs Max	The value shown is the maximum absolute value found between the cursors. If the maximum value was +1, and the minimum value was -1.5, then this mode would display 1.5.	
Peak to Peak	The value shown is the difference between maximum and minimum values found between the cursors. The peak to peak value is always positive.	
SD	The standard deviation from the mean of the values between the cursors. If there are no values between the cursors the field is blank. If there are n data points, and the sum of the squares of the differences between the points and the mean value is DiffSquare, the result is calculated as $\text{Sqrt}(\text{DiffSquare} / (n-1))$.	
RMS	The value shown is the RMS (Root Mean Square) of the values between the cursors. If there are no values between the cursors the field is blank.	
Peak	The value shown is the maximum value found between the cursors measured relative to a baseline formed by joining the two points where the cursors cross the data. This is always greater than or equal to 0.	
Trough	The value shown is the minimum value found between the cursors measured relative to a baseline formed by joining the two points where the cursors cross the data. This is always less than or equal to 0.	
RMS error	This is the same as the SD except that the normalising factor is n, not n-1.	
Mean in X	This works for waveforms and result views. The value is the mean x value weighted by the data at each point. Although this would normally be used with positive data, it has some meaning with negative data values.	
SD in X	This is available for result views only. The result has a useful meaning only when all the data values in the time range are positive. The value is: $\text{Sqrt}(\text{Sum}(\text{data}[x] * (x-x_m)^2) / \text{Sum}(\text{data}[x]))$ where the sum is over the x positions in the range and data[x] is the data value at x and x_m is the value returned by Mean in X.	
Mean Abs	This is the mean of the absolute values of the data values in the time range.	

Selecting, copying and printing

Fields from this window can be copied to the clipboard or directly to the Log window. To do this, select the region to be copied and then click the right mouse in the window and use the **Copy** or **Log** command. To select an entire row or column, click the mouse in the titles at the top and left of the window. To extend a selection, hold down the Shift key. To select non-contiguous rows or columns hold down the control key and select the rows and columns as required. You can also print the entire window with **Ctrl+P** and set the font with **Ctrl+F**.

Event channels

After version 5.06, event channels drawn in mean and instantaneous frequency mode are treated the same as waveforms. For events drawn in other modes, all measurement types except those listed below produce a blank field in the window. There is an **Edit menu Preferences** option in the **Compatibility** tab to force the old

behaviour, in which only the following measurement types produced a result:

Mean	The count of events or markers between the cursors divided by the time difference between the cursors.
Area	The total number of events or markers on the channel between the cursors.
Area (scaled)	The same as Area, but if a zero region is specified, the amount subtracted from the other regions is scaled by the relative width of the regions.
Sum	The same as Area.

Horizontal cursors

In many respects, horizontal cursors are similar to vertical cursors. However, they differ significantly in that each horizontal cursor is attached to a channel, or more accurately, to the y axis of a channel. If you change the drawing mode of an event channel with a horizontal cursor to a mode that has no y axis, the horizontal cursor will treat the available vertical space as if it runs from 0 at the bottom to 1 at the top.

If you drag channels with horizontal cursors on top of each other, only the horizontal cursor for the topmost channels (i.e. the channel with a valid y axis) is visible. In the special case of channels drawn with a locked y axis and a group offset of 0, all horizontal cursors for all channels are visible because the y axis is valid for all the channels.

The following commands in the **Cursor** menu let you control and interact with the horizontal cursors. There are also horizontal cursor context commands available by right-clicking on a horizontal cursors.

New Horizontal

The command is available when a time or result window is the current window, and there are less than 9 horizontal cursors already active. A new horizontal cursor is added to the first visible channel with a y axis. The cursor is given the lowest available horizontal cursor number and is labelled with the cursor label style for the window. If the x axis scroll bar is visible you can also activate this command by clicking the small button at the left of the x axis scroll bar.

You may find it more convenient to use the **Alt+1** through **Alt+9** keyboard shortcuts to activate horizontal cursors.

Delete Horizontal

The delete command activates a pop-up menu from which you can select a horizontal cursor to remove, or you can delete all the horizontal cursors. The cursors are listed with their number, channel number and position as an aid to identification. Deleting a cursor removes it from the window; other cursors are not affected.

Fetch Horizontal

This activates a drop down list from which you can select an existing horizontal cursor to move it to the first visible channel that has a Y axis. The vertical position in the channel is determined by the cursor number. It is usually much easier to use the keyboard shortcuts **Alt+1** through **Alt+9** to add a specific horizontal cursor to the first suitable channel.

Move To Horizontal

This command opens a pop-up menu from which you can select a horizontal cursor to move to. The cursors are listed with their number and position as an aid to identification. The window is scrolled vertically to display the nominated cursor in the centre of the y axis range.

Position Horizontal

This command opens a pop-up menu in which you can choose a horizontal cursor and then opens a dialog in which you can set the position and channel for the cursor. You can also open the dialog by right clicking on a horizontal cursor and selecting **Set Position** from the cursor context menu.

Display all Horizontal

This has no effect if there are no horizontal cursors. If there is a single cursor, the command behaves as though you had used the **Move To Horizontal** command and selected it. When there are multiple cursors, the channel y axis range is adjusted such that the lowest cursor is at the bottom edge of the channel area and the highest is at the top edge.

Horizontal Label mode

Each cursor has an optional label used to identify it. You can drag the cursor labels to the left and right with the mouse to suit the data. There are five cursor label modes: **None**, **Position**, **Number**, **Position and Number**, and **User-defined**. You select the most appropriate for your application using the pop-up menu or by right clicking on a cursor and choosing to set the cursor label from the context menu. To avoid confusion between the cursor number and the position, the number is displayed in bold type when it appears alone and bracketed with the position.

Each cursor stores its own mode and label, and the view has a mode that is applied to new cursors. The first four items in the menu set the view mode and the mode of all cursors. You can also set a user-defined label for cursors (but not for the view) with the **Set Label** command (which is the same as for the Vertical cursor).

Renumber Horizontal

When you create a horizontal cursor, they take the lowest available cursor number and you can also drag cursors over each other. In some circumstances you want the cursors numbered in order on the screen, for example, when taking the differences of cursor values. This command renumbers the cursors, with cursor 1 at the bottom. Where cursors share the same channel, they are ordered so that the lower numbered cursor has the lower y axis value.

Cursor context commands

If you right-click on a horizontal or vertical cursor, a context menu opens. This holds commands that can be applied to items in the view that are close to the mouse position. In particular, it holds the item **Cursor n** for vertical cursor n or **HCursor n** for horizontal cursor n. Move the mouse pointer over one of these commands to open a new context menu.

This menu contains commands to set the cursor position, set the cursor label, copy the cursor value to the clipboard and delete the cursor. For vertical cursors, you can open the Active cursor mode dialog for that cursor. Horizontal cursors also have the option to copy the cursor position relative to any of the other horizontal cursors.

12: Sample menu

Sample menu

The sampling menu controls the sampling configuration, and can start, stop and pause data capture. To capture data you need a CED 1401 interface (a Micro1401 or a Power1401). You can also capture data from other devices using the Talker interface. See the Sampling data chapter for more sampling information.


Sampling configuration

This command opens the Sampling Configuration dialog that sets the sampling parameters used when you select the File menu New command. You can load and save the configuration from the File menu Save Configuration and Load Configuration commands. See the Sampling data chapter for details.

Clear configuration

This command deletes all sampling window positions, associated result views for on-line processing, display trigger settings and WaveMark templates. It preserves the channel lists and all values visible in the Sampling Configuration dialog. If you want to reset the information in the Sampling Configuration dialog (list of channels, sample mode...), use the Reset button in the Sampling configuration dialog Channels Tab.

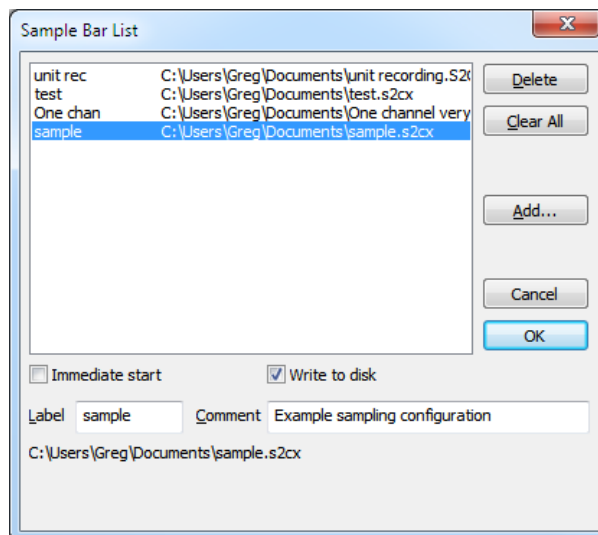
Sample bar

You can show, hide and manage the Sample Bar from the Sample  analysis inthABtr inthLast arbwave menu. You can show and hide it by right-clicking on any toolbar or on the Spike2 background. The Sample bar is a dockable toolbar with up to 20 user-defined buttons. Each button is linked to a Spike2 configuration file. When you click a button or use Alt+key (defined in the label), the associated configuration file loads and a new data file opens for sampling. Right-click a button to modify the button behaviour, open the associated configuration in the Sampling configuration dialog, remove the button from the bar or to open the Sample Bar List dialog.

The Sample menu Sample Bar List... command opens the Sample List dialog from where you can control the contents of the Sample bar.

The Add button opens a file dialog to select one or more Spike2 configuration files (*.s2cx or *.s2c) to add to the bar. If a file holds a label or comment, it is used, otherwise the first 8 characters of the file name form the label and the comment is blank.

Select an item in the list to edit the label and comment; this does not change the configuration file. If the label includes an ampersand (&), the next character becomes a short-cut key. To re-order buttons in the bar, drag items in the list. Delete removes the selected list item. Clear All deletes all items. Spike2 saves the Sample bar state in the registry. Each logon to Windows has its own registry settings, so each has its own Sample Bar settings.



Immediate start

Check this to sample immediately on a click of the selected Sample bar button rather than wait for the Sample control toolbar Start button. This control is also available by right-clicking on the buttons in the Sample bar.

Write to disk

If you have checked the Immediate start box you can also control whether the Write setting of the Sample control toolbar is checked when sampling begins. If you have not checked Immediate start, this has no effect.

Conditioner Settings

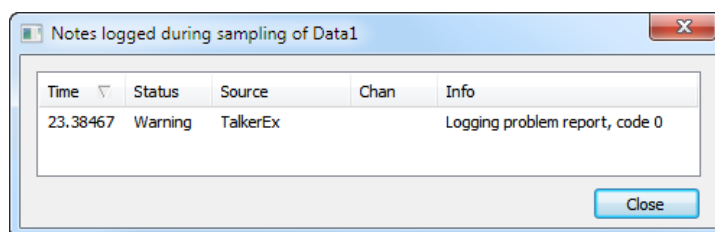
Spike2 supports several programmable signal conditioners. These devices amplify and filter waveform signals, and can provide other specialist functions. If a suitable conditioner is installed in your system, this menu command opens the conditioner dialog, see the *Programmable signal conditioners* chapter.

Sampling controls

This command hides and shows the Sample control toolbar. The controls within this toolbar are duplicated in this menu as the **Start sampling**, **Write to disk**, **Abort sampling** and **Reset sampling** commands. This toolbar is normally made visible (if hidden) when you open a data file for sampling. There is an **Edit Preference** to prevent this control bar being made visible every time you open a window for sampling. There is more information in the *Edit menu* chapter.

Sampling Notes

The Sample menu Sampling Notes command opens the resizable Notes dialog that lists Warning and Error messages that were generated either by a 1401 or a Talker during sampling. The list of messages is cleared each time sampling starts. The number of notes in the table is also displayed in the Sample Status bar.



If you right-click on the dialog you can access three context commands:

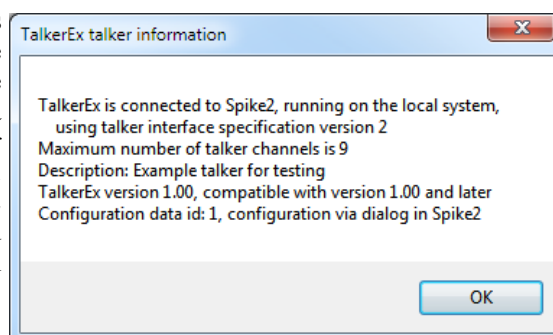
- Copy** Copy the selection to the clipboard as text
- Copy for spreadsheet** Copy the selection in a format suitable for pasting into a spreadsheet. Each field is surrounded by double quotation marks and fields are separated by Tab characters.
- Select All** Select all lines. To select sub-sections of the table: click to select a single line, **Shift+click** to select from the last selection and **Ctrl+click** to add the current line to the selection.

Talker list

The Sample menu Talkers command opens a pop-up menu that lists all Talkers that are currently connected, or that Spike2 has seen in the past and has not been told to forget. The Talker name in the list is followed by an exclamation mark (!) if the talker is not currently connected and available for sampling. If you select a talker name in the list, a further pop-up menu lists the following options:

Info...

This opens up the Talker Information Dialog. This lists information about the talker, including the talker interface specification version, current connection status, the maximum number of channels of data that are available, a brief description of the talker, the talker version number and any talker compatibility information that is available. Talkers that support the interface specification version 2 allow use of the `TalkerReadStr()` and `TalkerSendStr()` script commands. For example, with the TalkerEx connected on the local system:



Disconnect

This command will temporarily disconnect the Talker from Spike2.

Forget

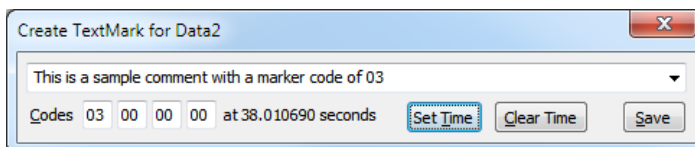
Spikes maintains a list of all talkers that it has seen so that you can set a sampling configuration without the Talker connected. The **Forget** option deletes all information about a Talker; you must reconnect to use it again. You cannot forget a connected Talker. See the *Sampling data* chapter for more about Talkers.

Sequencer controls

This command hides and shows the sequencer control panel that is available during sampling if an output sequence is active. There is more information about the sequencer control panel in the *Sampling data* chapter.

Create a TextMark


When you are sampling data, this menu command (short cut **Ctrl+T**) opens the Create TextMark window. To use this command you must enable a TextMark channel in the Sampling Configuration. You can always add markers by hand even if you enabled serial line input of TextMark data in the Sampling Configuration. The text of the last 10 markers is saved in the drop down list for reuse in a repeating protocol.



You can set the four marker codes to either a single printing character or to two hexadecimal digits.

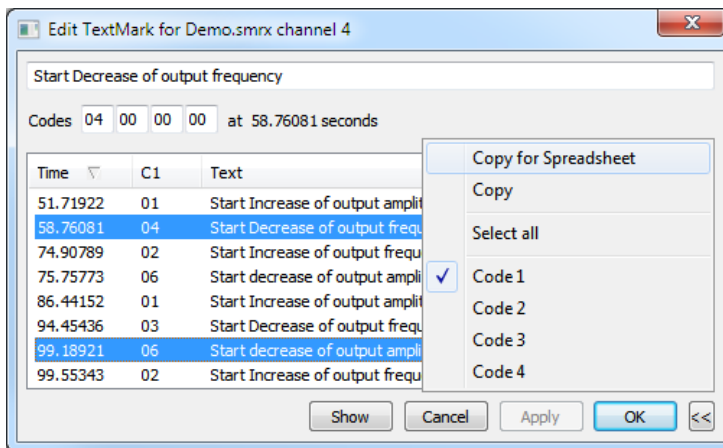
You commit the comment to the file when you press the **Save** button. The **Set Time** button fixes the time associated with the comment, the **Clear Time** button clears the time. If you have not set a time, the time of the comment is the time when it was saved.

If this channel is a trigger for Triggered sampling mode in the sampling configuration dialog, the trigger happens when you click the **Save** button and the **Save Time** and **Clear Time** buttons are replaced by the message **This channel is a trigger**.

 Comments are shown in the file as a small rectangle. The colour of the rectangle depends on the first marker code associated with the marker. If the code is 00, the rectangle is yellow. Otherwise, codes use the same colours as WaveMark data, set by the View menu **Change Colours** command.

You can see the text associated with a TextMark by moving the mouse pointer over the rectangle and waiting a moment. The text appears as a tool tip. Any movement of the mouse pointer hides the text.

To edit the comment text and marker codes, double click the rectangle to open the **Edit TextMark** dialog.



The marker codes display in the format set for the source channel (Hex only or using printing characters). Regardless of the channel format, you can always set the code in either format (two hexadecimal digits or one printing character).

The **>>** and **<<** buttons at the bottom right of the dialog show and hide the list area. Double-click an item in the list to edit it and save any changes to the item that was previously displayed. The **Show** button sets the time view time range to span the currently selected items. If you have made multiple selections, the first marker is displayed at the left edge of the screen and the last one at the right edge.

Right-click in the list to control how many marker codes to display and to copy data to the clipboard. If you choose **Copy for spreadsheet**, text strings and marker codes are enclosed in quotation marks so that the data will import easily into spreadsheet programs. You can change the sort order of the data in the list by clicking the column titles. Initially, the items are sorted based on the **Time** column.

Large numbers of TextMark items

The list box is limited to 12000 items, centred on the time of the item double clicked to open the dialog. If you work with a large number of TextMark items in a 32-bit .smr file, do not open the file across a network; network access is typically much slower than local file access and some operations, such as sorting the TextMark data items (by clicking on the column titles in the dialog) can be slow on large, networked files.

Change Output Sequence

This command is used during data sampling to replace the current output sequence. It displays a File Open dialog in which you can select an output sequence file. This file will replace the current sequence.

This will fail if there is no current sequence, or if the sequence or table size is larger than the current size or the size reserved in the Sequencer tab of the Sampling configuration dialog. If a sequence is replaced, it runs from the first instruction. Any variables that are declared with initial values are updated as are all initialised table values. The Sampling configuration is not updated with the name of this file.

Offline waveform output

When a data file is active, the Sample menu Waveform Output... command opens the Waveform output dialog. The script language equivalent of the majority of this dialog is the `PlayOffline()` command.

Channels to output

You can play up to 4 waveform or WaveMark channels through the 1401 DACs, or up to two channels through a sound card. To output multiple channels, select them in the time view, otherwise you can select single channels. You can change the channel selections after you open this dialog. Spike2 fills gaps in waveform data and gaps between WaveMark data items with zeros.

Time range

The Start time and End time fields set the file time range to play. You can link these to cursors in the time view. The values used are those set when you click Play.

To

You can select either a 1401 DAC for each channel, or if you select Sound in the first drop down list, the first (and second) channel is played through the sound card (mono for one channel, stereo for two channels).

Output sampling Rate

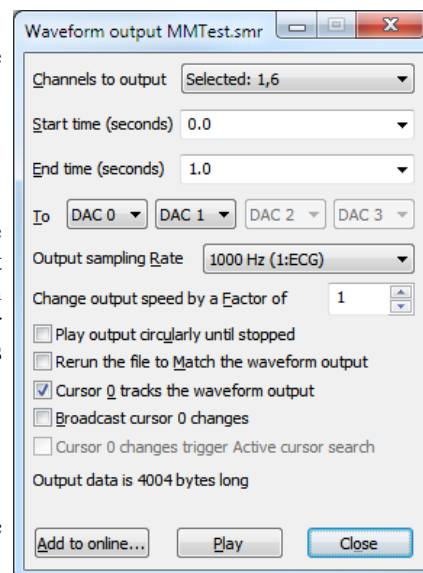
You can choose the sampling rate of any of the channels for output and the output device will get as close to this rate as it can. The 1401 sets the rate based on dividing down a 10 MHz clock; the rates achievable through a sound card depend on the card. All channels are replayed at the same speed. If any channel has a different sample rate, Spike2 resamples the data to have the same rate as set by this field (using linear interpolation).

Change output speed by a Factor of

You can speed up or slow down output by a factor of up to 4. This will change the pitch of sound or the repeat rate of a repetitive signal. The actual output rate depends on the capabilities of the output device.

Play output circularly

If you do not check this box, the waveform plays once only.



Rerun the file to Match the waveform output

If this box is checked when you click Play, the file will Rerun (simulate sampling) with the file time taken from the play position. This can be very useful when using Spike2 in teaching or for presentations, especially when using sound card output.

Cursor 0 tracks the waveform output

Check this box to have cursor 0 track the current playing position. Beware that cursor 0 has other uses in Spike2, for example in active cursors, and these may interfere with tracking the waveform replay position.

Broadcast cursor 0 changes

If you check the *Cursor 0 tracks the waveform output* box, this field is enabled. If you check it, each change made to cursor 0 for tracking purposes will be broadcast to all linked dialogs and to any multimedia windows associated with the time view. This will cause any open Cursor Values or Cursor Regions dialog box to update, which may make your system slow down.

Cursor 0 changes trigger Active cursor searches

If the *Broadcast cursor 0 changes* box is enabled and checked, this field is enabled. Each change to cursor 0 will cause an active cursor search.

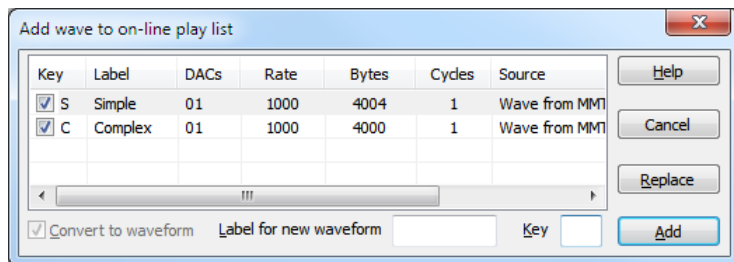
Play and Stop

This button starts output (and changes to Stop, which stops it). The output pays no attention to the scale and offset values associated with the channel. The 16-bit data values recorded for the file are played out. If you have an 8-bit sound card and try to play a low level signal, the result may be disappointing (or just silence).

Add to online...

This option opens a new dialog that adds the selected waveforms to the on-line waveform output list.

Add to online



This option adds the selected waveforms to the on-line waveform output list so that they can be replayed during data acquisition. The data file must have been saved to disk. You are allowed up to 10 waveforms. As long as the list of waveforms is not full and the current waveform is less than 32 MB in size you can Add the waveform to the list. You

can also Replace an existing waveform in the list. You can give each waveform a short name that labels the buttons for interactive replay of waveforms during data acquisition. There are more controls for the list of waveforms for replay in the Sampling Configuration dialog in the Play waveform tab.

Waveforms can be saved as either a data file, time range and a list of channels or by converting the current channel list into the memory image that would be played through the output DACs on line. If a waveform has been converted the Source column in the list starts with "Memory". The advantages of converting are:

- The output is not affected if the original file is moved or deleted or changed
- The (usually small) time for the conversion is saved each time you sample data

However, there are also disadvantages to converting:

- It can take a lot of memory to hold the converted data which can slow Spike2 down and may cause your system to run out of memory
- If the sampling configuration is saved, the converted data is also saved, which makes the configuration files much larger and can slow down system operation.

To prevent the system running out of memory, there is a limit on the amount of memory (32 MB) that any one converted wave may use. If any channel in the play list is a WaveMark or a memory channel or a duplicate

channel, Spike2 will always attempt to convert the data, as there is no guarantee that these channels will exist (or have the same marker filter) when sampling is requested.

If a channel has a channel process applied to it, you must either leave the data file open with the channel process applied or you must convert the wave. This is because a channel process only has effect while the data file is open in Spike2.

13: Script menu

Script menu

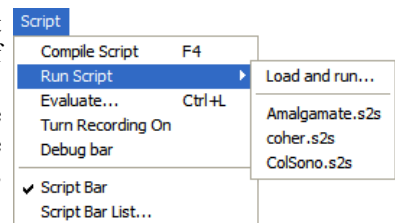
The script menu gives you access to the scripting system. From it you can compile a script, run a loaded script, evaluate a script command for immediate execution and record your actions as a script. Spike2 version 8 removes the option to convert a script from the format used in the DOS version of Spike2; if you still need this facility you must use Spike2 version 7.

Compile Script

This option is enabled when the current window holds a script. It is equivalent to the compile button in the script window. Spike2 checks the syntax of the script, and if it is correct, it generates a compiled version of the script, ready to run.

Run Script

This option pops-up a list of all the scripts that have been loaded so that you can select a script to run. Spike2 compiles the selected script and if there are no errors, the script runs. If you run a script twice in succession, Spike2 compiles it for the first run, and uses the compiled result for the second run, saving the compilation time. If a script stops with a run time error, the script window is brought to the front and the offending line is highlighted.



You can also select the Load and Run... option from which you can select a script to run. The script is hidden and run immediately (unless a syntax error is found in it).

Evaluate

This Script menu command opens the evaluate window where you can type in and run one line scripts. You can cycle round the last 10 script lines with the buttons. The Execute button compiles and runs your script; Eval(...) does the same, and also displays the value of the last expression on the line in the evaluate bar and also in the Log view.

Turn Recording On Off

You can record your actions into the equivalent script. Use this command to turn recording on and off. When you turn recording off, a new script window opens that holds the commands that are equivalent to your actions. When recording is enabled REC appears in the status bar.

Caveats

Note that not all actions are recordable and that there is no guarantee that a recorded script will run. Some actions depend on the overall state of the system and this may well not be the same as when the script was recorded. That said, the majority of actions do record and the recorded script can often be used as the skeleton for a script. We do make some attempts to not record actions that have no effect. For example, if you drag a window around, we only record the final position before you perform some other action; we do not record every intermediate position.

Created channels and views

If you record actions that create new views or memory, virtual or duplicate channels, we assign variables to hold the new view handles and channel numbers. We do this because for these actions, the new channel number or view handle may not be the same if you repeat the action. For example, if you record the action of creating a memory channel and then set the channel comment, the recorded script is:

```
var v12% := ViewFind("Demo.smr");
```

```
var ch1%; 'MemChan created channel
FrontView(v12%);
ch1% := MemChan(2,0,0,0);
ChanShow(ch1%); 'Make it visible
ChanComment$(ch1%, "Comment for recording example");
```

By recording the generated memory channel in a variable, the `ChanShow()` and `ChanComment$()` commands can act on the new channel, regardless of the number it was assigned. If you do not create a channel as part of the recording, it will be referred to by the channel number, not by a variable. When you stop recording, all knowledge of recorded channel numbers is cleared.

Timing

Recording does not take account for how long you take to do something. For example, if you record the actions: open new data document, start sampling, wait 100 seconds, stop sampling, the recorded script will not have the 100 second wait. If you want to include this you will need to edit the result. To insert time delays you can use a toolbar or dialog idle routine or the `Yield()` command.

Debug bar

You can show and hide the debug bar from this menu when the current view is a script. You can also show and hide the debug bar by right clicking on the title bar of the Spike2 window, or on the application window.

Script bar

You can show and hide the Script Bar and manage the Script Bar contents from the Script menu. You can also show and hide the Script Bar by clicking the right mouse button on any Spike2 toolbar or on the Spike2 background. The Script Bar is a dockable toolbar with up to 20 user-defined buttons. Each button is linked to a Spike2 script file. When you click a button, the associated script file is loaded and run. There is also a user-defined comment associated with each button that appears as a tool-tip when the mouse pointer lingers over a button.

You can right-click on a button to open a context menu from where you can open the script in the script editor, remove the button from the bar or open the Script Bar List dialog.

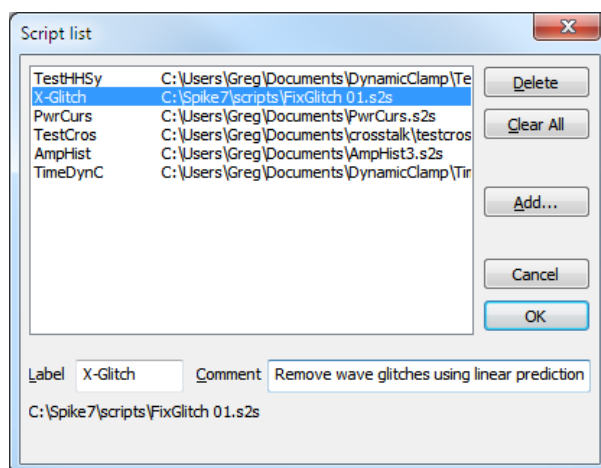
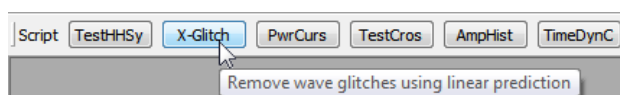
The Script menu `Script Bar List...` command opens the Script List dialog from where you can control the contents of the Script Bar.

The Add buttons opens a file dialog in which you can choose one or more Spike2 script files (*.s2s) to add to the bar. If the first line of a script starts with a single quote followed by a dollar sign, the rest of the line is interpreted as a label and a comment, otherwise the first 8 characters of the file name form the label and the comment is blank. The label is separated from the comment by a vertical bar. The label is be up to 8 characters (and can include an ampersand for keyboard activation, see below) and the comment up to 80 characters. A typical first line might be:

```
'$DlgMake|Write script for a simple dialog box
```

You can select an item in the list and edit the label and comment. This does not change the contents of the script file. You can re-order buttons in the bar by dragging items in the list. The `Delete` button removes the selected item. `Clear All` removes all items from the list.

Spike2 saves the Script bar state in the registry when it closes and loads it when it runs. Each different logon to Windows has its own registry configuration, so if your system has three users, each has their own Script bar settings. Alternatively, you can have different experimental configurations by logging on as a different user name. The script language `ScriptBar()` function has the same functionality as this dialog.



Keyboard short-cuts for Script and Sample bars

You can activate a script in the Script bar from the keyboard. To do this, add an ampersand (&) to the script label before the keyboard character you want to use to run the script. The key is not case sensitive, so &x and &X are equivalent. Then, as long as the script bar is visible and there is not a script already running and there is not a modal dialog open (a dialog that requires you to close it before you can do anything else), you can type `Alt+keyboard character` to simulate clicking the script button.

You must be aware that any key you use overrides the use of that key by the menu system as you get first use of any key. For example, if you set a label to "&FixChan", you have disabled the use of the keyboard to access the File menu.

The Sample bar also supports `Alt` key activation, but the script bar has higher priority. To activate a Sample bar configuration, the Sample bar must be visible, there must be no open sampling document and a script must not have disabled the Sample menu. Keys defined for the Script bar take precedence over Sample bar keys.

14: Help menu

Help menu

Spike2 has comprehensive Help built in and usually accessible with the F1 key or by clicking on a Help button. The Help menu gives you some control over the help system and also contains the About Spike2 command.

Using help

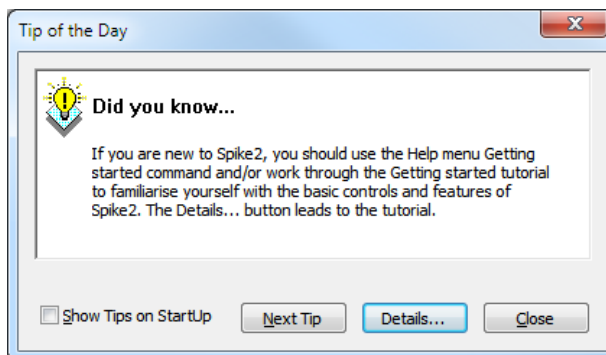
Spike2 supports context sensitive help that duplicates the contents of this manual and the script language manual. You can activate context sensitive help with the F1 key from most dialogs to get a description of each field in the dialog. You can use the help Search dialog to lookup topics that are not covered by the index.

In a script window you can obtain help by placing the cursor on any keyword in the script and pressing F1. To get help on a script function, type the function name followed by a left hand bracket, for example MemChan(, then make sure that the cursor lies to the left of the bracket and in the function name and press F1.

The help system has indexes, hypertext links, keyword searches, help history, bookmarks and annotations. If you are unsure of using help, once you have opened the help window, use the Help menu Using help command for detailed instructions.

Tip of the Day

You can show the Tip of the Day window when Spike2 starts by checking the box. Of course, if you hate this sort of thing, clear the check box and you never need see it again! If the Details... button is enabled, it opens the on-line Help at a page that holds more information about the topic in the window. The tips are held in the hidden file sonview.tip in the same folder as Spike2. If you have a useful tip that you would like to share with others, send it to us and we will very likely include it in the next release.



View Web site

If you have an Internet browser installed in your system, this command will launch it and attempt to connect to the CED web site (www.ced.co.uk). The site contains down-loadable scripts, updates to Spike2 and information about CED products.

Getting started

This command runs a demonstration script that gives a quick tour of Spike2 basics. The option is disabled if the script is not present or if you are sampling data.

Other sources of help

If you are having trouble using Spike2, please do the following before contacting the CED Software Help Desk:

1. Try looking in the on-line help system for more information. Use Search to find related topics.
2. If the problem is in sampling data, run the Try1401 program provided as part of Spike2. This checks out the 1401, device driver and interface.

If none of the above helps, FAX, email (softhelp@ced.co.uk) or call the CED Software Help Desk. Please include a problem description, Spike2 serial number and program version and a description of the circumstances leading to the problem. It would also help us to know the type of computer you use, how much memory it has and which version of your operating system you are running.

About Spike2

The About Spike2 command is found in the Help menu. It opens an information dialog that contains the serial number of your licensed copy of Spike2, plus your name and organisation. The dialog also display information about the 1401 device drive and your 1401. If you contact us for assistance we will need the information in this dialog. The Copy button writes the Spike2 version and serial number, your user name and organisation (that was entered when you installed Spike2), the device driver and 1401 information and your working set size to the clipboard. You can then paste this into an email or other text document.



Spike2 version information

The top line of text identifies the version of Spike2 that you are running, the target platform (x86 for a 32-bit build or x64 for a 64-bit build) and the date when we built the code. This is followed by our copyright statement. The third line is present if Spike2 can contact the CED website. If there is a more recent version available, there will be a message of the form: Version 8.nn is available for download. If your version is up to date, we display: Spike2 is up to date. If we could not contact the site, or if the information we read makes no sense, the line is blank. If there is an update available you can click the Web Site button go to our general update page to collect it.

1401 device driver

If there is a 1401 device driver installed, the driver revision is displayed. If the driver is older than Spike2 expects, you will be warned. Spike2 displays the type of 1401 and the monitor version if a 1401 is connected and powered up. If you have installed Spike2 correctly, you should not have any problems as Spike2 comes with the latest 1401 drivers available at the time we generated the release. If you really need to update the drivers independently of Spike2 you can follow the Device drivers link on our general update page.

1401 firmware revision

If the 1401 monitor or 1401 firmware is not the most recent at the time this version of Spike2 was released, an asterisk follows the version. If it is so old that it compromises data sampling, two asterisks follow the version.

The Power1401 and Micro1401 mk II and -3 have firmware in flash memory. Flash updates and instructions for applying them are available as downloads from the CED web site; you can update the flash firmware without opening the 1401 case. There are downloads for: Power1401-3, Power1401 mk II, Power1401 mk I, Micro1401 mk II, Micro1401-3.

To fix this, you should follow the link for your 1401, go to the download and follow the detailed instructions to update your 1401. The update is done by the Try1401 program that is installed at the same time as Spike2, so you already have all the tools you need to complete the task.

1401: Monitor or firmware too old or interface problem

This message appears if we detect that the 1401 monitor ROM is too old to run Spike2 safely or if there is a problem with 1401 communications such that the 1401 is detected but communications are garbled.

Working set size

There is information about the Working Set Size at the bottom of the About box. The two numbers describe the minimum and maximum physical memory that the operating system allows Spike2 to use when memory is running low. If you suffer from error -544 when you sample data, these numbers are important. See the

Technical Support chapter for more information.

15: Script language

Script language

Introduction to scripting

For many users, the interactive nature of Spike2 may provide all the functionality required. This is often the case where the basic problem to be addressed is to present the data in a variety of formats for visual inspection with, perhaps, a few numbers to extract by cursor analysis and a picture to cut and paste into another application. However, some users need analysis of the form:


1. Find the peak after the second stimulus pulse from now.
2. Find the trough after that.
3. Compute the time difference between these two points and the slope of the line.
4. Print the results.
5. If not at the end, go back to step 1.

This could all be done manually, but it would be very tedious. A script can automate this process, however it requires more effort initially to write it. A script is a list of instructions (which can include loops, branches and calls to user-defined and built-in functions) that control the Spike2 environment. You can create scripts by example, or type them in by hand.

Hello world

Traditionally, the first thing written in every computer language prints “Hello world” to the output device. Here is our version that does it twice! To run this, use the File menu New command and select Script Document, then click OK to create a new, empty script window. Type or copy the following into the script window:

```
Message("Hello world");  
PrintLog("Hello world");
```

Click the  Run button to check and run your first script. If you have made any typing errors, Spike2 will tell you and you must correct them before the script will run. The first line displays “Hello world” in a box and you must click on OK to close it. The second line writes the same text to the Log view. Open the Log view to see the result (if the Log window is hidden you should use the Window menu Show command to make it visible).

So, how does this work? Spike2 recognises names followed by round brackets as a request to perform an operation (called a *function* in computer-speak). Spike2 has around 400 built-in functions, and you can add more with the script language. You pass the function extra information inside the round brackets. The additional information passed to a function is called the function *arguments*.

Spike2 interprets the first line as a request to use the function `Message()` with the argument "Hello world". The message is enclosed in double quotation marks to flag that it is to be interpreted as text, and not as a function name or a variable name.

An argument containing text is called a *string*. A string is one of the three basic data types in Spike2. The other two are *integer* numbers (like 2, -6 and 0) and *real* numbers (like 3.14159, -27.6 and 3.0e+8). These data types can be stored in *variables*.

Spike2 runs your script in much the same way as you would read it. Operations are performed in reading order (left to right, top to bottom). There are also special commands you can insert in the script to make it run round loops or do one operation rather than another. These are described in the script language syntax chapter.

Spike2 can give you a lot of help when writing a script. Move the text caret to the middle of the word `Message` and press the F1 key. Help for the `Message()` command appears, and at the bottom of the help entry you will find a list of related commands that you might also find useful.

Views and view handles

The most basic concept in a script is that of a view and the view handle that identifies it. A view is a window in Spike2 that the script language can manipulate. The running script is hidden from most commands, however you can obtain its handle using `App(3)` so you can show and hide it.

There is always a *current view*. Even if you close all windows the Log view, used for text output by the `PrintLog()` command, remains. Whenever you use a built-in function that creates a new view, the function returns a *view handle*. The view handle is an integer number that identifies the view. It is used with the `View()` and `FrontView()` functions to specify the current view and the view that should be on top of all windows.

Whenever a script creates a new view, the view becomes the current view. However, views are created invisibly so that they can be configured before appearing. You can use `WindowVisible(1)` to display a new window.

You can see a list of view handles by using the **Window** menu **Windows...** command. When debugging a script, you can see the current view handle by opening the Global variables window.

Writing scripts by example

To help you write scripts Spike2 can monitor your actions and write the equivalent script. This is a great way to get going writing scripts, but it has limitations. Scripts generated this way only repeat actions that you have already made. The good point of recording your actions is that Spike2 shows you the correct function to use for each activity.

For example, let us suppose you use the **Script** menu **Turn Recording On** option, open a data file, select interval histogram analysis mode of channel 3, process all data in the file and end with the **Script** menu **Stop recording** command. Spike2 opens a new window holding the equivalent script (we have tidied the output up a little and added comments):

```
var v6%,v7%;          'declare integer variables to hold view handles
v6% := FileOpen("demo.smr", 0, 3);    'Open file, save view handle
Window(10, 10, 80, 50);              'Set a window position
v7% := SetInth(3, 100, 0.01, 0);      'Create invisible INTH view
WindowVisible(1);                    'make the INTH visible
Process(0, View(-1).Maxtime(), 0, 1); 'Add data to INTH view
```

Some of this is fairly straightforward. You can find the `FileOpen()`, `SetInth()`, and `Process()` functions described in this manual and they seem to map onto the actions that you performed. However, there is extra scaffolding holding up the structure.

In the first line, the `var` keyword creates two integer variables, `v6%` and `v7%`. These variables hold view handles returned by `FileOpen()` and `SetInth()`. Spike2 generates the names from the internal view numbers (so your script may not be exactly the same). The result of the functions is copied to the variable with the `:=` *assignment operator*. In English, the second line of the script could be read as: *Variable v6% is assigned the result of the FileOpen command on file demo.smr.*

The `SetInth()` command makes a new window to hold an interval histogram of the data in channel 3 with 100 bins of 0.01 seconds width and with the first bin starting at an interval of 0 seconds. The `WindowVisible(1)` command is present because the new window created by `SetInth()` is hidden. Spike2 creates invisible windows so that you can size and position them before display to prevent excessive screen repainting.

The `View(-1).` syntax accesses data belonging to a view other than the current view. The current view when the `Process()` command is used is the result view and we want to access the maximum time in the original time view. The negative argument tells the script system that we want to change the current view to the time view associated with this result view. The dot after the command means that the swap is temporary, only for the duration of the `Maxtime()` function.

Using recorded actions

You can now run this script with the Run button at the top right of the script window. The script runs and generates a new result view, repeating your actions. Now suppose we want to run this for several files, each one selected by the user. You must edit the script a bit more and add in some looping control (`while...wend`). The following script suggests a solution. Notice that we have now changed the view handle variables to names that are a little easier to remember.

```
var fileH%, inthH%;           'view handle variables
fileH% := FileOpen("", 0, 1); 'blank for dialog, single window
while fileH% > 0 do          'while fileH% is greater than 0
  inthH% := SetInth(3, 100, .01, 0); WindowVisible(1);
  Process(0, View(-1).Maxtime(), 0, 1);
  Draw();                    'Update the INTH display
  fileH% := FileOpen("", 0, 1); 'ask for the next file
wend;                        'The end of the while loop
```

This time, Spike2 prompts you for the file to open. The file identifier is negative if anything goes wrong opening the file, or if you press the **Cancel** button. We have also included a `Draw()` statement to force Spike2 to draw the data after it calculates the interval histogram. There is a problem with this script if you open a file that does not contain a channel 3 that holds events or markers of some kind. We will deal with this a little later.

However, you will find that the screen gets rather cluttered up with windows. We do not want the original window once we have calculated the histogram, so the next step is to delete it. We also have added a line to close down all the windows at the start, to reduce the clutter when the script starts.

```
var fileH%, inthH%;
FileClose(-1);                'close all windows except script
fileH% := FileOpen("", 0, 1); 'use a blank name to open dialog
while fileH% > 0 do          'FileOpen returns -ve if no file
  inthH% := SetInth(3, 100, .01, 0); WindowVisible(1);
  Process(0, View(-1).Maxtime(), 0, 1);
  View(fileH%).FileClose();  'Shut the old window
  Draw();                    'Update the INTH display
  fileH% := FileOpen("", 0, 1); 'ask for the next file
wend;
```

This seems somewhat better, but we still have the problem that there will be an error if the file does not hold a channel 3, or it is of the wrong type. The solution to this is to ask the user to choose a channel using a dialog. We will have a dialog with a single field that asks us to select a suitable channel:

```
var fileH%, inthH%, chan%;   'Add a new variable for channel
FileClose(-1);                'close all windows to tidy up
fileH% := FileOpen("", 0, 1); 'use a blank name to open dialog
while fileH% > 0 do          'FileOpen returns -ve if no file
  DlgCreate("Channel selection"); 'Start a dialog
  DlgChan(1, "Choose channel for INTH", 126); 'all but waveform
  if (DlgShow(chan%) > 0) and 'User pressed OK and...
    (chan% > 0) then          '...selected a channel?
    inthH% := SetInth(chan%, 100, .01, 0); WindowVisible(1);
    Process(0, View(-1).Maxtime(), 0, 1);
    View(fileH%).FileClose(); 'Shut the old window
    Draw();                    'Update the display
  endif
  fileH% := FileOpen("", 0, 1); 'ask for the next file
wend;
```

The `DlgCreate()` function has started the definition of a dialog with one field that the user can control. The `DlgChan()` function sets a prompt for the field, and declares it to be a channel list from which we must select a channel (or we can select the No channel entry). The `DlgShow()` function opens the dialog and waits for you to select a channel and press **OK** or **Cancel**. The `if` statement checks that all is well before making the histogram.

Differences between systems

If you need to write code that can run on different systems, you should try to avoid system dependent features, or if this is impossible, make use of the `System()` function to find out where you are. For example, the Macintosh versions of Spike2 stopped at version 2, so if you use any Spike2 version 3 or later features, your script will not run on a Macintosh.

Notation conventions

Throughout this manual we use the font of this sentence to signify descriptive text. Function declarations, variables and code examples print in a monospaced font, for example `a := View(0)`. We show optional keywords and arguments to functions in curly braces:

```
Func Example(needed1, needed2 {,opt1 {,opt2}});
```

In this example, the first two arguments are always required; the last two are optional. Any of the following would be acceptable uses of the function:

```
a := Example(1,2);           'Call omitting the optional arguments
a := Example(1,2,3);        'Call omitting one argument
a := Example(1,2,3,4);      'Call using all arguments
```

A vertical bar between arguments means that there is a choice of argument type:

```
Func Choice( i%|r|str$ );
```

In this case, the function takes a single argument that could be an integer, a real or a string. The function will detect the type that you have passed and may perform a different action depending upon the type.

Three dots (...) stand for a list of further, similar items. It is also used when a function can accept an array with any number of dimensions:

```
Func Sin(x|x[]{|...});
```

This means that the function `Sin()` will accept a real value or an array with one or more dimensions.

Other sources of script information

This continues with information about the script window and debugging, followed by a reference for the script language and then documentation for the built-in script functions, first with functions grouped by topic, and then a full alphabetical list.

When you are in the script editor, there are several features designed to help you write script code. These include pop-up help tips, right-clicks to take you to the help for a specific function, pop-up help when adding arguments to functions, automatic formatting and Auto-complete of typed names.

There are example and utility scripts provided with Spike2. These are copied to the `scripts` folder within the folder that contains Spike2.

Our web site at www.ced.co.uk has example scripts and script updates that you can download.

There is also a manual that has been used for our Spike2 training courses, held at CED and around the world. This Training Day manual contains many annotated examples and tutorials. Some of the scripts in this manual are useful in their own right; others provide skeletons upon which you can build your own applications. You can get a PDF of this manual from our web site on the Downloads page (follow the links Software manuals, Spike2 V8, Training).

Script window and debugging

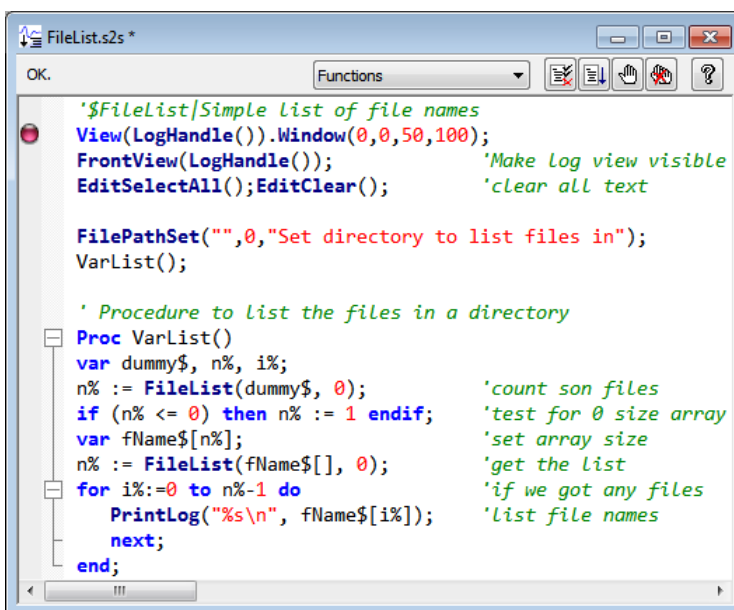
You use the script window when you write and debug a script. Once you are satisfied that your script runs correctly you would normally run a script from the script menu without displaying the source code. You can have several scripts loaded at a time and select one to run with the Script menu Run Script command.

Script editor

If a script is read from a read only medium or is write protected you cannot make changes to it in the Spike2 script editor.

The script window is a text window with a few extra controls including a folding margin that allows you to fold away inner loops, function and procedures. The folding margin can be hidden; see the Edit menu Preferences for details of configuring the script view folding margin.

To the left of the text area is a margin where you can set break points (one is shown already set), bookmarks and where the current line of the script is indicated during debugging. Above the text area is a message bar and several controls. The controls have the following functions:



This control is a quick way to find any `func` or `proc` in your script. Click on this to display a list, in alphabetical order, of the names of all user-defined routines. Select one, and the window will scroll to it. To be located, the keywords `func` and `proc` must be at the start of a line and the name of the routine must be on the same line.

Compile

The script compiler checks the syntax of the script and if no errors are found it creates the compiled version, ready to run. If the script has not been changed since the last compile and no other script has been compiled, the button is disabled as there is no need to compile again. Spike2 can have one compiled script in memory at a time.

Run

If the script has not been compiled it is compiled first. If no errors are found, Spike2 runs the compiled version, starting from the beginning. Spike2 skips over `proc ... end;` and `func ... end;` statements, so the initial code can come before, between or after any user-defined procedures and functions. This button is disabled once the script has started to run.

Set break point

This button sets a break point on the line containing the text caret or clears a break if one is already set. A break point stops a running script when it reaches the start of the line containing the break point. You can also set and clear break points by clicking the mouse pointer in the margin or by right clicking the line and using the Toggle break command. A break point is indicated by a red marker in the gutter (between the optional line number and folding margins). You can set and clear break points by clicking in the gutter, or using the context

menu (right-click) in a script view.

Not all statements can have break points set on them. Some statements, such as `var`, `const`, `func` and `proc` compile to entries in a symbol table; they generate no code. If you set a break point on one of them the break point will appear at the next statement that is “breakable”. If you set break points before you compile your script, you may find that some break points move to the next breakable line when you compile.

Once a script has been compiled, all breakpoints in it and in any included files are remembered. You can close and open the files and the break points will still be visible. However, break points are reassessed when you click the run button to start a script. This means that if you want to break in an included file you must have the file open and the break point set when you hit run. Once the script has started to run you can close any or all of the script files and break points will still work and automatically open the source file when they are hit.



Clear all break points

This button is enabled if there are break points set in the script. Click this button to remove all break points from the script. Break points can be set and cleared at any time, even before your script has been compiled.



Help

This button provides help on the script language. It takes you to an alphabetic list of all the built-in script functions. If you scroll to the bottom of this list you can also find links to the script language syntax and to the script language commands grouped by function. Within a script, you can get help on keywords and built-in commands by clicking in the keyword or command and pressing the `F1` key or by right-clicking on a name and selecting **Help for name** from the context menu.

You can also get a pop-up call tip to appear if you hover the mouse pointer over a user-defined or built-in `PROC` or `FUNC` name.

Go to user-defined Proc or Func

You can navigate to the start of a user-defined `PROC` or `FUNC` by right-clicking on the **name** and selecting **Go to name** from the context menu. This works, even if the named item is defined in an included file (as long as the included file can be located).

Syntax colouring

Spike2 supports syntax colouring for both the script language and also for the output sequencer editor. You can customise the colouring (or disable it) from the Script files settings section of the **Edit menu Preferences** in the **General** tab. The language keywords have the standard colour blue, quoted text strings have the standard colour red, and comments have the standard colour green. You can also set the colour for normal text (standard colour black) and for the text background (standard colour white).

The syntax colouring options are saved in the Windows registry. If several users share the same computer, they can each have their own colouring preferences as long as they log on as different users.

Editing features for scripts

There are some extra editing features that can help you when writing scripts. These include automatic formatting, commenting and un-commenting of selected lines, indent and outdent of code, code folding, auto-complete of typed words and pop-up help for built-in and user-defined functions. These are described in the documentation for the **Edit** menu.

Debug overview

Despite all our efforts to make writing a script easy, and all your efforts to get things right, sooner or later (usually sooner), a script will refuse to do what you expect. Rather than admit to human error, programmers attribute such failures to “bugs”, hence the act of removing such effects is “debugging”. The term dates back to times when an insect in the high voltage supply to a thermionic valve really could cause hardware problems.

To make bug extermination a relatively simple task, Spike2 has a “debugger” built into the script system. With the debugger you can:

- Step one statement at a time
- Step into or over procedures and functions
- Step out of a procedure or function
- Step to a particular line
- Enter the debugger on a script error to view variable values
- View local and global variables
- Watch selected local and global variables
- Edit variable values
- See how you reached a particular function or procedure
- Set and clear break points

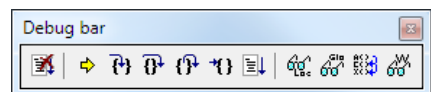
With these tools at your disposal, most bugs are easy to track down.

Preparing to debug

A Spike2 script does not need any special preparation for debugging except that you must set a break point or include the `Debug()` command at the point in your script where you want to enter the debugger. Alternatively, you can enter the debugger “on demand” by pressing the `ESC` key; you may need to hold it down for a second or two, depending on what the script is doing. If the `Toolbar()`, `DlgShow()` or `Interact()` commands are active, hold down `ESC` and click on a button. This is a very useful if your script is running round in a loop with no exit! You can disable this with the `Debug(0)` command, but we suggest that this feature is added after the script has been tested! Once you have disabled debugging, there is no way out of a loop.

You can also choose to enter debug on a script error by checking the **Enter debug on script error** box in the **General** tab of the **Edit menu Preferences** option. Depending on the error, this may let you check the values of variables to help you fix the problem.

When your script enters the debugger, the debug toolbar opens if it was not visible. The picture shows the toolbar as a floating window, but you can dock it to any side of the Spike2 window by dragging it over the window edge and releasing.



Stop running the script (`Ctrl+Alt+Shift+F5`). There is no check that you really meant to do this; we assume that if you know enough to open the debugger, you know what you are doing! You can use the `Debug()` command to disable the debugger.



Display the current script line (`Ctrl+Alt+F2`). If the script window is hidden, this makes it visible, brings it to the top and scrolls the text to the current line.



If the current statement contains a call to a user-defined `PROC` or a `FUNC`, step into it, otherwise just step (`Ctrl+F5`). This does not work with the `Toolbar()` command which is not user-defined, but which can cause user-defined routines to be called. To step into a user-defined `FUNC` that is linked to a `Toolbar()` command, set a break point in the `FUNC`.



Step over this statement to the next statement (`Shift+F5`). If you have more than one statement on a line you will have to click this button once for each statement, not once per line.



Step out of a procedure or function (`Alt+F5`). This does not work if you are in a function run from the `Toolbar()` command as there is nowhere to return to. In this case, the button behaves as if you had pressed the run button.



Run to cursor, or more accurately to the start of the line with the text caret (`Ctrl+Alt+F2`). This is slightly quicker than setting a break point, running to it, then clearing it (which is how this is implemented). This button is disabled unless the current view is a script file that is part of the currently running script.



Run the script (`Ctrl+Shift+F5`). This disables the buttons on the debug toolbar and the script runs until it reaches a break point or the end of the script.



Show the local variables for the current user-defined `func` or `proc` (`F7`). If there is no current routine, the window is empty. You can edit a value by double clicking on the variable. Elements of arrays are displayed for the width of the text window. If an array is longer than the space in the window, the text

display for the array ends with ... to show that there is more data. You can right-click on a variable to add it to the Watch window.



Show the global variable values in a window (Ctrl+F7). You can edit a global variable by double clicking on it. The very first entry in this window lists the current view by handle, type and window name. You can right-click on a variable to add it to the Watch window.



Display the call stack (list of calls to user-defined functions with their arguments) on the way to the current line in a window (Ctrl+Shift+F7). If the Toolbar() function has been used, the arguments for it appear, but the function name is blank.



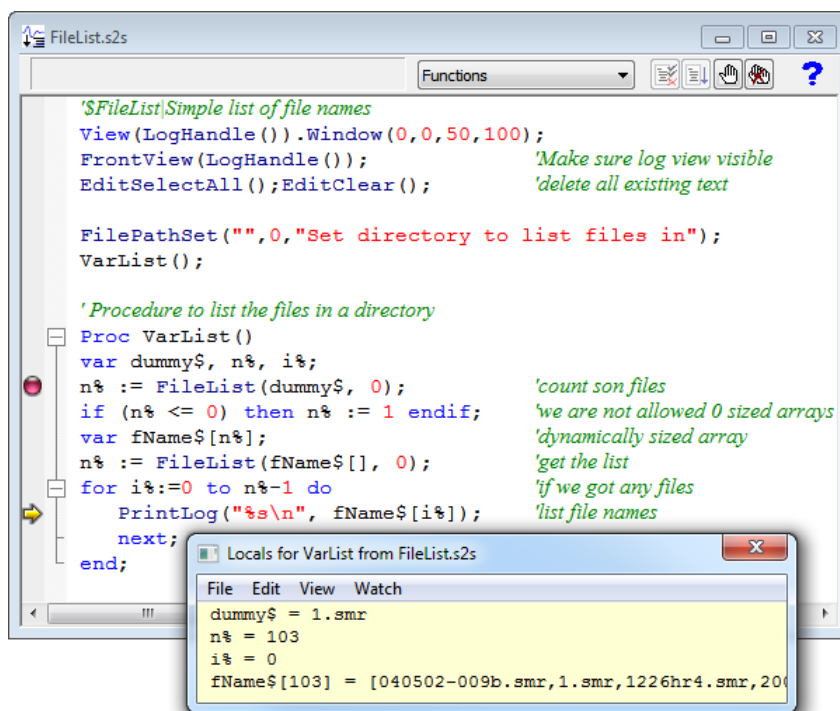
Open the Watch window. This displays globals and locals that were selected in the global or local variables windows. Local variables are displayed with values when the user-defined Proc or Func in which they are defined is active.

The debug toolbar and the locals, globals and the call window close at the end of a script. The buttons in the debug toolbar are disabled if they cannot be used. If you forget what a particular button does, move the mouse pointer over the button. A "Tool tip" window will open next to the button with a short description; if the Status bar is visible, a longer description can be seen there.

Enter debug on error

There is an option in the Edit menu Preferences dialog General tab that allows you to enter the debugger if an error occurs. After an error you can inspect the values of local and global variables and the contents of the call stack, but you are not allowed to continue running the script.

Inspecting variables



If the watch, locals or globals windows are open, they display a list of variables. If there are more variables than can fit in the window you can scroll the list up and down to show them all. Simple variables are followed by their values. If you double click on one a new window opens in which you can edit the value of the variable. Variables in the local and global windows are displayed in the order they are declared in the script. Variables in the watch window are displayed in the order that you add them, but you can use the Watch menu to sort them into alphabetic order.

If you double click on an array, a new window opens that lists the values of the elements of the array. You choose the element by editing the index fields, one for each dimension.

Menu commands

The variable windows contain a menu with the following commands:

File	Close	Closes the debug window
Edit	Copy	Copy selected text to the clipboard
	Log	Copy selected text to the Log view
	Select All	Select all windows text
	Find...	Open the Find text window
	Find Again	Repeat the last find operation
	Find Last	Repeat last find operation searching backwards
	Toggle Bookmark	Set/clear bookmark in Global variable window
	Next Bookmark	Jump to the next bookmark in the Global variable window
View	Previous Bookmark	Jump to previous bookmark in the Global variable window
	Clear All Bookmarks	Remove all bookmarks from the Global variable window
View	Font	Open the Font dialog to change the window settings
Watch	Add to the Watch window	Add selected Globals and Locals items
	Delete from the Watch window	Remove selected Watch window item
	Delete all watched variables	Remove all items from the Watch window
	Delete 'Not found' variables	Remove watched items that are not in the current script
	Sort variables into alphabetic order	Sort Watch window items into alphabetic order

Most commands have keyboard shortcuts listed in the menu. The Watch menu items are also available on a right-click context menu, where appropriate.

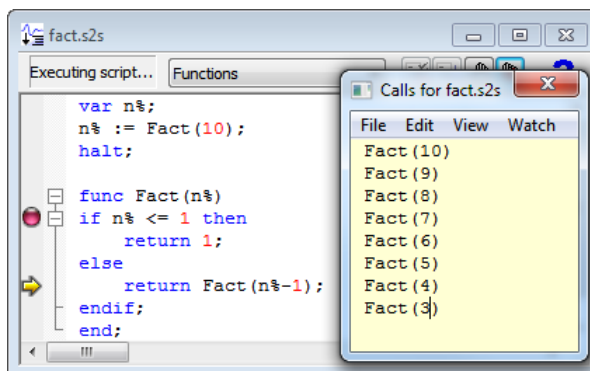
Watch window

You can add variables to the watch window by right-clicking on them in the locals or globals window and choose the option to copy the selected variables to the watch window. In the watch window, right-click to see available options to control the watched variables. The watch window remembers the watched variables between debugging sessions. If a variable does not exist in the current script, it is still remembered, but is marked as not existing.

Call stack

The call stack can sometimes be useful to figure out how your script arrived at a position in your code. This is particularly true if your script makes recursive use of functions. A function is recursive when it calls itself, either directly, or indirectly through other functions.

A tricky fault to diagnose is a script that has mutually recursive functions. If there is no way out, the script gets deeper and deeper into the call stack until it runs out of memory. The call stack can help to detect such problems. The example to the right demonstrates the use of a recursive function (but see `LnGamma()` and `BinomialC()` for large factorials)



Script language syntax

A Spike2 **script** is composed of lines of text. Each line can be up to 240 characters long (this is an arbitrary limit - most lines will be much shorter). Scripts are usually written in the script editor, but can be written with any editor and imported. The script uses 8-bit ASCII characters.

A script consists of program **statements**. Statements that generate looping or branching constructs include

other statements within them. A statement is terminated by a semicolon or by a keyword that is part of an enclosing statement. Statements are not terminated by an end of line character, so a statement can be spread over multiple lines. Two semicolons in a row generate an empty statement, which generates no code.

Within a statement, **white space** of any length is treated as a single space character, and white space between script tokens is ignored unless it is required to separate two tokens. White space consists of the space and tab characters and end of line characters.

A **token** is an indivisible entity such as a script keyword, the name of a constant, variable, procedure or function or a numeric or string constant. White space can be added anywhere without changing the meaning of a script except in the middle of a token.

Keywords and names

All keywords, user-defined functions and variable names in the script language start with one of the letters a to z followed by the characters a to z and 0 to 9. Keywords and names are not case sensitive, however users are encouraged to be consistent in their use of case as it makes scripts easier to read. Variables and user-defined functions use the characters % and \$ at the end of the name to indicate integer and string type.

User-defined names can extend up to a line in length. Most users will restrict themselves to a maximum of 20 or so characters.

The following keywords are reserved and cannot be used for variables or function names:

```
and   band   bor     break  bxor   case
const continue diag   do     docase else
end   endcase endif  for    func   halt
if    mod     next   not    or     proc
repeat resize  return step  then   to
trans until   var    view   wend   while
xor
```

Further, names used by Spike2 built-in functions cannot be redefined as user functions or global variables. They can be used as local variables (not recommended).

Data types

There are three basic data types in the script language: real, integer and string. The real and integer types store numbers; the string type stores characters. Integer numbers have no fractional part, and are useful for indexing arrays or for describing objects for which fractions have no meaning. Integers have a limited (but large) range of allowed values.

Real numbers span a very large range of number and can have fractional parts. They are often used to describe real-world quantities, for example the weight of an object.

Strings hold text and automatically grow and shrink in length to suit the number of text characters stored within them.

Real data type

This type is a double precision floating point number. Variables of this type have no special character to identify them. Real constants have a decimal point or the letter e or E to differentiate from integers. White space is not allowed in a sequence of characters that define a real number. Real number constants have one of the following formats where digit is a decimal digit in the range 0 to 9:

```
{-}{digit(s)}digit.{digit(s)}{e|E}{+|-}digit(s)}
{-}{digit(s)}.digit{digit(s)}{e|E}{+|-}digit(s)}
{-}{digit(s)}digitE|e{+|-}digit(s)}
```

Real numbers have a precision of more than 16 decimal digits. The maximum magnitude that can be stored is 1.7976931348623158e+308 and the minimum non-zero magnitude is 2.2250738585072014e-308. The smallest number that can be added to 1.0 so that the result is not 1.0 is 2.2204460492503131e-016. A number must fit on a line and be within the range of acceptable values. There is no limit on the number of

digits after the decimal point. However, supplying more than 17 digits of precision has no effect. The following are legal real numbers:

```
1.2345 -3.14159 .1 1. 1e6 23e-6 -43e+03
```

E or e followed by a power of 10 introduces exponential format. The last three numbers above are: 1000000 0.000023 -43000.0. The following are not real constants:

1 e6	White space is not allowed	1E3.5	Fractional powers are not allowed
2.0E	Missing exponent digits	1e500	The number is too large

Technical stuff

Real numbers in Spike2 are stored in the double format defined in IEEE 754-1985, which is a standard format used in most computers. Each number uses 64 bits, of which 11 hold an exponent, there is one sign bit, and 52 bits hold the magnitude of the data, making 64 bits. The format manages to squeeze one extra bit out of the magnitude by stating that all normalised numbers start with a 1 bit that need not be supplied.

Unlike integer numeric expressions, which are exact unless you overflow their value range, the precision of a calculation involving real numbers may be problematic. Multiplication and division do not cause any loss in precision other than a rounding effect in the least significant bit of the result. However, addition and subtraction can cause significant loss. The expression: $1e15 + (1.0/3.0) - 1e15$ is not evaluated as 0.3333333... as you might hope, but 0.375 and $1e16 + (1.0/3.0) - 1e16$ has the value 0. When taking the difference of numbers of similar sizes, the precision of the result is limited by the precision of the larger number. You can sometimes reduce the precision loss by rearranging expressions.

When comparing real numbers that are the results of calculations, do not test for exact equality. For example, the following will never end:

```
var a := 0.0;
repeat
  a += 1/999.0;   'Beware: 1/999 evaluates as 0 as integer/integer -> integer
until a = 1.0;
```

The problem is that 1/999.0 cannot be represented exactly in the real number format, the closest that can be stored is 0.00100100100100100100; this is slightly less than the exact result. It is easy to see that after adding this up 999 times, the result is 0.9999999999999999900, which is not 1. Changing the last line to:

```
until a >= 1.0;
```

guarantees that the loop will terminate.

Real numbers can store integral values exactly in the range -9007199254740992 to 9007199254740992 (this is 2 to the power 53). Ceil(), Floor(), Trunc() and Round() have no effect on real numbers greater in magnitude than 9007199254740992 as they have no fractional part.

Integer data type

The integer type is identified by a % at the end of the variable name and stores 64-bit signed integer (whole) numbers in the range -9223372036854775808 to 9223372036854775807. There is no decimal point in an integer number. An integer number has the following formats (where digit is a decimal digit 0 to 9, and hexadecimal-digit is 0 to 9 or a to f or A to F, with a to f standing for decimal 10 to 15):

```
{-}{digit(s)}digit
{-}0x|X{hexadecimal-digit(s)}hexadecimal-digit
```

You may assign real numbers to an integer; it is an error to assign numbers beyond the integer range. Real numbers have any fractional part removed before assignment (-1.99 is -1 as an integer). If you assign an integer to a real number you can lose accuracy as real numbers have typically 53 bits of precision. Put another way, if you assign an integer larger than 9007199254740992 ($0x2000000000000000$, 9×10^{15}) to a real number you will lose precision. When differencing very large integers, use integer arithmetic for maximum accuracy.

The following are examples of integers:

```
1 -1 -2147483647 0 0x6789abcd 0x100 -0xcd
```

Integer limits and older versions of Spike2

Feature	64-bit	32-bit
---------	--------	--------

Decimal range	-9223372036854775808 to 9223372036854775807	-2147483648 to 2147483647
Hexadecimal range	0x8000000000000000 to 0x7fffffffffffffff	0x80000000 to 0x7fffffff
As a power of 10	-9.223×10^{18} to 9.223×10^{18}	-2.147×10^9 to 2.147×10^9
Size in bytes	8	4

Before Spike2 version 8 the script language used 32-bit integers. If you wish your script to run in version 7 you must limit your use of integers to the 32-bit range. Note that times in the 64-bit filing system are 64-bit values, so if you read event times using `ChanData()` you can read values that exceed the 32-bit integer range.

String data type

Strings are lists of characters. String variable names end in a `$`. String variables can hold strings up to 65534 characters long. Literal strings in the body of a program are enclosed in double quotation marks, for example:

```
"This is a string"
```

A string literal may not extend over more than one line. Consecutive strings with only white space between them are concatenated, so the following:

```
"This string starts on one lin"
"e and ends on another"
```

is interpreted as "This string starts on one line and ends on another". Strings can hold special characters, introduced by the escape character backslash:

```
\ " The double quote character (this would normally terminate the string)
\\ The Backslash character (beware paths, use: "C:\\nope\\top.ext" or "C:/nope/top.ext")
\t The Tab character
\n The New Line character
\r The Carriage Return character (ASCII code 13)
```

Conversion between data types

You can assign integer numbers to real variables and real numbers to integer variables (unless the real number is out of the integer range when a run-time error will occur). When a real number is converted to an integer, it is truncated. You can use the `Round()`, `Trunc()`, `Floor()` and `Ceil()` functions to convert floating point values to integral floating point values. The `Asc()`, `Chr$()`, `Str$()`, `Print$()`, `ReadStr()`, and `Val()` functions convert between strings and numbers.

Global and local variable declarations

Variables are created by the `var` keyword. This is followed by a list of variable names and optional initialising expressions and the list is terminated by a semicolon. You must declare all variable names before you can use them. Arrays are declared as variables with the size of each dimension in square brackets. The first item in an array is at index 0. If an array is declared as being of size `n`, the last element is indexed `n-1`.

```
var myInt%,myReal,myString$;      'an integer, a real and a string
var aInt@[20],ar1[100],aStr@[3]  'integer, real and string vectors
var a2d[10][4];                  '10 rows of 4 columns of reals
var square@[3][3];               '3 rows of 3 columns of strings
```

You can define variables in the main program, or in user-defined functions. Those defined in the main program are *global* and can be accessed from anywhere in the script after their definition. Variables defined in user-defined functions and procedures exist from the point of definition to the end of the function or procedure and are deleted when the function ends. We refer to these as *local* variables. If you have a recursive function, each time you enter the function you get a fresh set of variables.

Before version 7.11, the dimensions of global arrays had to be constant expressions (but see `resize`). The dimensions of local arrays can be set by variables or calculated expressions. Simple variables (not arrays) can be initialised when they are declared. Uninitialised numeric variables are set to 0; uninitialised strings are empty.

```
var Sam%:=3+2, pi:=4*ATan(1), sal$:="I am \"Sally\"";
```

Compatibility with previous versions of Spike2

Before Spike2 version 7, the initialising expression could not include variables or function calls. If you want to write a script that is compatible with version 6, the previous example must be written as:

```
var Sam%:=3+2, pi, sal$:="I am \"Sally\"";  
pi := 4*ATan(1);
```

Constant declarations

Constants are created by the `const` keyword. A constant can be of any of the three basic data types, must be initialised in the declaration with a constant expression and cannot be an array. A constant expression is composed of previously defined numeric constants, numbers and the operators add, subtract, multiply and divide (+-*/) or is a string constant. The syntax and use of constants is the same as for variables, except that you cannot assign to them or pass them to a function or procedure as a reference parameter.

```
const Sam%:=3+2, pi:=3.1415926535897932, sal$:="I am \"Sally\"";
```

Arrays of data

The three basic types (integers, reals and strings) can be made into arrays with from 1 to 5 dimensions. Before Spike2 version 6, the maximum number of dimensions allowed was 2). We call a one-dimensional array a vector, and a two-dimensional array a matrix to match common usage. Declare arrays with the `var` statement:

```
var v[20], M[10][1000], nd[2][3][4][5][6], text$[23];
```

This declares a vector with 10 elements, a matrix with 10 rows and 1000 columns, a 5-dimensional array with 720 elements and a one-dimensional array of strings. To reference array elements, enclose the element number in square brackets (the first element in each dimension is number 0):

```
v[6] := 1; x := M[8][997]; nd[1][0][0][0][2] := 4.5; text$[1]:="a";
```

You can declare an array with one or more dimensions set to 0! However, such an array has limited utility; you can `resize` it, it can be passed to user-defined `proc` and `func` and some built-in script functions will accept 0-sized arrays (and do nothing) but in most cases, all dimensions must be non-zero to use the array.

There is a maximum number of elements (product of the sizes of the dimensions) that you are allowed when you declare an array using `var`. This is currently set to 100,000,000 in an attempt to prevent operations that would likely take a very long time. If you need more elements than this, declare the array with a 0-sized dimension, then use `resize` (which we do not limit), or declare the size using a non-constant expression (which the compiler cannot check). However, be prepared for the script to fail if you run out of allocatable memory, which can happen, even if the size is less than the limit.

You will find that above some size, arrays will become slow when accessed randomly. This happens when the operating system decides that your application is using too much memory and pages some out to disk, then loads it in as you use it. If you access such memory sequentially, the effect is not too disastrous as you will be loading new pages every few hundred accesses. However, if you access memory randomly, you will likely trigger a disk read every access. A disk read takes of order milliseconds compared to a fraction of a microsecond for a memory access.

Prior to Spike2 version 7.11, dimension sizes for an array declared outside a `Proc` or `Func` (a global array) were all constants; from version 7.11 they can be defined with a variable expression. Inside a `Proc` or `Func` the sizes can be set by variables. For example:

```
Proc VariableSizeArray(n%)  
var x[n%];  
...
```

You cannot have two global `var` statements that refer to the same variable. That is, you cannot have code like:

```
var fred[23][32];  
...  
var fred[23][48];           'This line will generate an error
```

as this will generate an error: "Identifer is already defined at 'fred' ". In a `Proc` or `Func`, you

can declare an array inside a loop, and change the size of the dimensions each time around the loop. However, since version 7 we urge you to declare arrays outside loops and use `resize` to do any required size changes.

```

Proc BadStyle()
var i%;
for i% := 1 to 100 do
    var arr[i%];           'this has always been allowed...
    ...
next;
end;

Proc BetterStyle()
var arr[0], i%;           'declare the array once
for i% := 1 to 100 do
    resize arr[i%];       'resize it
    ...
next;
end;

```

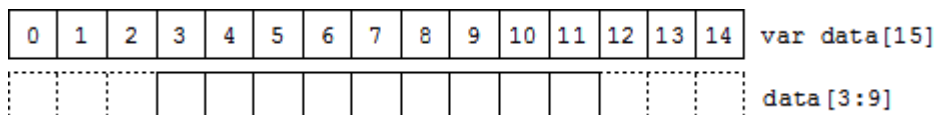
We may make resizing an array using `var` illegal in the future, so please do not do it in new code. Resizing with `var` preserved the original data when the last dimension is changed, but changes to any other dimension do not preserve the data.

Vector subsets

Use `v[start:size]` to pass a vector or a subset of a vector `v` to a function. `start` is the index of the first element to pass, `size` is the number of elements. Both `start` and `size` are optional. If you omit `start`, 0 is used. If you omit `size`, the sub-set extends to the end of the vector. To pass the entire vector use `v[0:]`, `v[:]`, `v[]` or just `v`.

For example, consider the vector of real numbers declared as `var data[15]`. This has 15 elements numbered 0 to 14. To pass it to a function as a vector, you could specify:

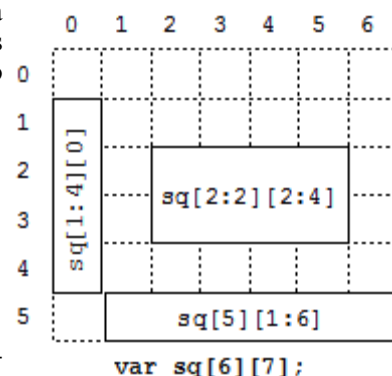
<code>data</code> or <code>data[]</code>	This is the entire vector. This is the same as <code>data[:]</code> or <code>data[0:15]</code> .
<code>data[3:9]</code>	This is a vector of length 9, being elements 3 to 11.
<code>data[:8]</code>	This is a vector of length 8, being elements 0 to 7.



Matrix subsets

With a matrix you have more options. You can pass a single element, a vector sub-set, or a matrix sub-set. Consider the matrix of real numbers defined as `var sq[6][7]`. You can pass this as a vector or a matrix to a function as (`a`, `b`, `c` and `d` are integer numbers):

<code>sq[a][b:c]</code>	a vector of length <code>c</code>
<code>sq[a][]</code>	a vector of length 7
<code>sq[a:b][c]</code>	a vector of length <code>b</code>
<code>sq[][c]</code>	a vector of length 6
<code>sq[a:b][c:d]</code>	a matrix of size <code>[b][d]</code>
<code>sq</code> or <code>sq[][]</code>	a matrix of size <code>[6][7]</code>



This diagram shows how sub-sets are constructed. `sq[1:4][0]` is a 4 element vector. This could be passed to a function that expects a vector as an argument. `sq[5][1:6]` is a vector with 6 elements. `sq[2:2][2:4]` is a matrix, of size `[2][4]`.

N-Dimensional array subsets

With more than 2 dimensions, you can make a subset of any number of dimensions up to the size of the original array. These examples show some of the possibilities for passing a source array with 5 dimensions defined as `var nd[4][5][6][7][8]`;

<code>nd</code> or <code>nd[][][][]</code>	The entire 5 dimensional array
<code>nd[0][][][]</code>	A 4 dimensional array of size <code>[5][6][7][8]</code>
<code>nd[1:2][][]</code>	A 4 dimensional array of size <code>[2][5][7][8]</code>

```
nd[1][2][3][4][ ]    A vector of size [8]
nd[ ][ ][0][0][0]    A matrix of size [4][5]
```

Transpose of an array

You can pass the transpose of a vector or matrix to a function with the `trans()` operator, or by adding ``` (back-quote) after the vector or matrix name. The transpose of a matrix swaps the rows and columns. To be consistent with matrix mathematics, a one-dimensional array is treated as a column vector and is transposed into a matrix with 1 row. That is given `var data[15]`, `trans(data)` is a matrix of size `[1][15]`.

```
var M[5][3], v[5], W[5][5];
PrintLog(M, M`);           'Print M and its transpose
PrintLog(M[ ][ ], trans(M[ ][ ])); 'Exactly the same as last line
MatMul(W, M, M[ ][ ]`);    'set W to M times its transpose
MatMul(W, v, v`);          'set W to v times its transpose
```

From Spike2 version 6 onwards you can apply the transpose operator to arrays of higher dimensions. The result is an array with the dimensions and indexing reversed. That is, the transpose of `x[2][3][4]` is an array of size `[4][3][2]`. The element `x[i][j][k]` in the original becomes the element at index `[k][j][i]` in the transposed array.

Diagonal of a matrix or array

You can pass the diagonal of a matrix to a function using the `diag()` operator. This expects a matrix as an argument and produces a vector whose length is the smaller of the dimensions of the matrix. Given a matrix `M[10][10]`, `diag(M)` is a 10 element vector.

From version 6 of Spike2 you can take the diagonal of any array with more than 1 dimension. The result is a vector with the length of the smallest dimension of the array. For example, given `var a[4][5][6]`, `diag(a)` is a vector of length 4 holding the elements: `a[0][0][0]`, `a[1][1][1]`, `a[2][2][2]` and `a[3][3][3]`.

Memory usage of arrays

Each element of a real array uses 8 bytes of memory. Each element of an integer array uses 4 bytes of memory and each element of a string array uses 16 bytes, plus the memory to hold the text in the string (one byte per character) plus a terminating 0. Each array is held in contiguous memory (except that each string element holds a pointer to the string text that is not in this contiguous memory). An array can require a lot of memory. For example, a real array with three dimensions of 1000, 100 and 10 has 1 million elements, each using 8 bytes. If you use many, large arrays, it is possible to run out of system memory. When this will happen is difficult to predict as it depends on many system settings, on how much physical memory your system has and how much virtual memory space the system will allocate to you. Once you exceed the limits of available physical RAM, random array access can become very slow as data is swapped between memory and disk. Adding more physical memory to your system can help if this is a problem. Spike2 scripts will stop with an error if you request more memory than the system can allocate to you.

Changes to arrays at version 7.11

Global arrays can be defined with a variable size. The following did not compile before version 7.11:

```
var n% := 10, x[n%];
```

You can now pass arrays of zero size to user-defined functions; previously this caused a fatal error. You can also pass zero-sized arrays to some built-in functions, for example `ArrConst()`.

Resize array

You cannot change the number of dimensions of an array, but you can change the size of the dimensions. This is done with the `resize` statement (added at version Spike2 version 7), which has a syntax that is very similar to `var`:

```
resize v[24], M[2][3000], nd[6][5][4][3][2], text$[923];
```

When used in this way, the values in the square brackets, which can be expressions or constants, set the new size for each dimension. However, if you want to leave a dimension at the current size, you can use:

```
resize nd[ ][ ][ ][ ][n%]; 'change last dimension only
```

A pair of empty square brackets means that you want to preserve the current size of the corresponding dimension. The `resize` statement preserves data in the array (unless you make one or more dimensions smaller, when data is omitted). When you make dimensions larger, new numeric array elements are set to 0; new string elements are set to an empty string ("").

Unlike `var`, which checks the total size of the array (unless the arguments are expressions), there is no compiler-imposed limit on the total array size. However, each individual dimension is limited (currently to 100,000,000 elements). The script will stop with a fatal error if memory cannot be allocated.

In most cases, you will only want to change one dimension to cope with adding more items to an array. It is more efficient to increase the last dimension as in this case it is often possible to extend (or reduce) the memory allocated to the array without physically moving it in memory. If you change any other dimension than the last, the `resize` statement allocates a new array of the required size, copies data into it, replaces the original array with the new one and releases the memory used by the original array.

You can always resize a global or local array unless a sub-array, transpose or diagonal of it has been passed to a Proc of Func and is currently in use. You will get the error message: "Attempt to index non-array or resize sub-array or result view" if you break this rule. Here are some examples to make this clearer:

```
var global[2][3];
Level1(global);           'pass entire global array
resize global[3][3];     'OK

proc Level1(g[[]])        'g is entire global array
var local[3][4];
TryResize(local);        'this is OK, passing entire array
TryResize(g);            'this is OK, passing entire array
TryResize(global);       'this is OK, passing entire array
TryResize(local[:2][]);  'will fail as is a sub-array
ObscureError(local[:2][]); 'pass sub-array of local, OK
ObscureError(g[:1][:1]); 'pass sub-array of global, not OK
end;

Proc TryResize(arr[[]])
resize arr[][2];
end;

Proc ObscureError(h[[]]) 'will fail in the resize...
resize global[4][];     '...if h is a sub-array of global
end;
```

When you create a sub-array, transpose or diagonal of an existing array, a temporary array construct is created that depends on the original. If you were to resize the original, all the dependant arrays that referred to it would become invalid, so we do not allow you to make such a change. You cannot resize an array derived from a result view.

Efficiency

If you are adding items to an array, it is very inefficient to increase the array size for each item added. Apart from being very slow, this will cause a pattern of memory allocation that is about the worst possible for the performance of the system. The standard solution in this case is to start with a reasonable size, one that will be big enough for most situations, then when you need more, to allocate a sensible extra portion of space. If you have no idea how big the target is, the best algorithm (best in terms of reducing the number of reallocations and memory fragmentation) is to double the size each time you run out. However, this is also the most wasteful of memory. Increasing by a fixed amount or a fixed proportion of the existing size may work. Do NOT increase by one each time unless the array is very small and is never going to get very big.

Result views as arrays

The script language treats a result view as vectors of real numbers, one vector per channel. To access a vector element use `View(v%, ch%).[index]` where `v%` is the view, `ch%` is the channel and `index` is the bin number, starting from 0. You can pass a channel as an array to a function using `View(v%, ch%).[]`, or `View(v%, ch%).[a:b]` to pass a vector subset starting at element `a` of length `b`. You can omit `ch%`, in which case channel 1 is used. You can also omit `View(v%, ch%)`, in which case channel 1 in the current view is used.

If you change a visible result view, the modified area is marked as invalid and will update at the next opportunity.

Memory usage of result views

Each channel uses 8 bytes per bin per channel. However, if you are averaging data and want to see the variance of each data point, this costs you an extra 8 bytes per bin, and if you want to keep individual bin counts, this costs a further 4 bytes per bin.

If you are saving event times into a channel for Raster display, the cost is around 100 bytes for each raster line and 8 bytes for every event.

If you average 20 channels of 50000 points per channel with all options on, this will use 20 MB of memory. This is not too bad in itself, but if you are generating a lot of result views and leaving them open, the system may start paging memory to disk, which will make things feel slow.

Statement types

The script language is composed of statements. Statements are separated by a semicolon. Semicolons are not required before `else`, `endif`, `case`, `endcase`, `until`, `next`, `end` and `wend`, or after `end`, `endif`, `endcase`, `next` and `wend`. White space is allowed between items in statements, and statements can be spread over several lines. Statements may include comments. Statements are of one of the following types:

- A variable or constant declaration terminated by a semicolon
- An assignment statement of the form:

<code>variable := expression;</code>	Set the variable to the value of the expression
<code>variable += expression;</code>	Add the expression value to the variable
<code>variable -= expression;</code>	Subtract the expression value from the variable
<code>variable *= expression;</code>	Multiply the variable by the expression value
<code>variable /= expression;</code>	Divide the variable by the expression value

The `+=`, `-=`, `*=` and `/=` assignments were added at version 3.02. `+=` can also be used with strings (`a$+=b$` is the same as `a$:=a$+b$`, but is more efficient).

- A flow of control statement, described below
- A procedure call or a function with the result ignored, for example `view(vh%);`

Comments in a script

A comment is introduced by a single quotation mark. All text after the quotation mark is ignored up to the end of the line.

```
view(vh%); 'This is a comment, and extends to the end of the line
```

Spike2 does not have a block comment marker that allows comments to start on one line and end on another.

Expressions and operators

Anywhere in the script where a numeric value can be used, so can a numeric expression. Anywhere a string constant can be used, so can a string expression. Expressions are formed from functions, variables, constants, brackets and operators.

Numeric operators

In numerical expressions, the following operators are allowed, listed in order of precedence:

	Operators	Names
Highest	<code>`</code> , <code>[]</code> , <code>()</code>	Matrix transpose, subscript, round brackets
	<code>-</code> , <code>not</code>	Unary minus, logical not
	<code>*</code> , <code>/</code> , <code>mod</code>	Multiply, divide and modulus (remainder)
	<code>+</code> , <code>-</code>	Add and subtract

	<, <=, >, >=	Less, less or equal, greater, greater or equal
	=, <>	Equal and not equal
	and, band	Logical and, bitwise and
	or, xor, bor, bxor	Logical or, exclusive or and bitwise versions
Lowest	?:	Ternary operator

The order of precedence determines the order in which operators are applied within an expression. Without rules on the order of precedence, $4+2*3$ could be interpreted as 18 or 10 depending on whether the add or the multiply was done first. Our rules say that multiply has a higher precedence, so the result is 10. If in doubt, use round brackets, as in $4+(2*3)$ to make your meaning clear. Extra brackets do not slow down the script.

The divide operator returns an integer result if both the divisor and the dividend are integers. If either is a real value, the result is a real. So $1/3$ evaluates to 0, while $1.0/3$, $1/3.0$ and $1.0/3.0$ all evaluate to $0.333333\dots$

The minus sign occurs twice in the list because minus is used in two distinct ways: to form the difference of two values (as in `fred:=23-jim`) and to negate a single value (`fred :=-jim`). Operators that work with two values are called *binary*, operators that work with a single value are called *unary*. There are four unary operators, `[]`, `()`, `-` and `not`, the remainder are binary.

There is no explicit `TRUE` or `FALSE` keyword in the language. The value zero is treated as false, and any non-zero value is treated as true. Logical comparisons have the value 1 for true. So `not 0` has the value 1, and the `not` of any other value is 0. If you use a real number for a logical test, remember that the only way to guarantee that a real number is zero is by assigning zero to it. For example, the following loop may never end:

```
var add:=1.0;
repeat
  add := add - 1.0/3.0; ' beware, 1/3 would have the value 0!
until add = 0.0;      ' beware, add may never be exactly 0
```

Even changing the final test to `add<=0.0` leads to a loop that could cycle 3 or 4 times depending on the implementation of floating point numbers.

The result of the comparison operators is integer 0 if the comparison is false and integer 1 if the comparison is true. The result of the binary arithmetic operators is integer if both operands are integers, otherwise the result is a real number. The result of the logical operators is integer 0 or 1. The result of the exclusive or operator is true if one operand is true and the other is false.

The bitwise operators `band`, `bor` and `bxor` treat their operands as integers, and produce an integer result on a bit by bit basis. They are not allowed with real number operands.

String operators

	Operators	Names
Highest	+	Concatenate
	<, <=, >, >=	Less, less or equal, greater, greater or equal
	=, <>	Equal and not equal
Lowest	?:	Ternary operator

The comparison operators can be applied to strings. Strings are compared character by character, from left to right. The comparison is case sensitive. To be equal, two strings must be identical. You can also use the `+` operator with strings to concatenate them (join them together). The character order for comparisons (lowest to highest) is:

```
space !"#%&'()*+,-./0123456789:;<=>?@
ABCDEFGHIJKLMNPOQRSTUVWXYZ[\]^_`
abcdefghijklmnopqrstuvwxyz{|}~
```

Do not confuse assignment `:=` with the equality comparison operator, `=`. They are entirely different. The result of an assignment does not have a value, so you cannot write statements like `a:=b:=c`; (which was allowed in the MS-DOS version of Spike2).

The ternary operator

The ternary operator `? :` was added to the script language at Spike2 version 6 and has the following format:

```
numeric expression ? expression1 : expression2
```

The result of the ternary operator is *expression1* if *numeric expression* evaluates to a non-zero result otherwise it is *expression2*. You can use this anywhere that an expression would be acceptable, including when passing arrays as arguments to functions. However, *expression1* and *expression2* must be type compatible in the context of their use. For example, if one is a string, then the other must also be a string. If they are arrays passed as arguments, they must have the same type and the same number of dimensions. If they are arguments passed by reference, they must have identical type. In an expression, one can be integer and the other real, in which case the combined type is treated as real.

You are not allowed to use this operator to choose between function or procedure names passed as arguments into functions or procedures.

Examples of expressions

The following (meaningless) code gives examples of expressions.

```
var jim,fred,sam,sue%,pip%,alf$,jane$;
jim := Sin(0.25) + Cos(0.25);
fred := 2 + 3 * 4;           'Result is 14.0 as * has higher precedence
fred := (2 + 3)* 4;         'Result is 20.0
fred += 1;                   'Add 1 to fred
sue% := 49.734;              'Result is 49
sue% := -49.734;             'Result is -49
pip% := 1 + fred > 9;        'Result is 1 as 21.0 is greater than 9
jane$ := pip% > 0 ? "Jane" : "John"; 'Result is "Jane"
alf$ := "alf";
sam := jane$ > alf$;          'Result is 0.0 (a is greater than J)
sam := UCase$(jane$)>UCase$(alf$); 'Result is 1.0 (A < J)
sam := "same" > "sam";       'Result is 1.0
pip% := 23 mod 7;            'Result is 2
jim := 23 mod 6.5;           'Result is 3.5
jim := -32 mod 6;            'Result is -2.0
sue% := jim and not sam;     'Result is 0 (jim = -2.0 and sam = 1.0)
pip% := 1 and 0 or 2>1;      'Result is 1
sue% := 9 band 8;            'Result is 8 (9=1001 in binary, 8=1000)
sue% := 9 bxor 8;            'Result is 1
sue% := 9 bor 8;             'Result is 9
```

Mathematical constants

We don't provide maths constants as built-in symbols, but the two most common ones, *e* and *pi* are easily generated within a script; *e* is `Exp(1.0)` and π (*pi*) is `4.0*ATan(1.0)`. Alternatively, here are their values to as much accuracy as you can store in a double :

```
 $\pi$       3.141592653589793
e       2.718281828459045
```

Flow of control statements

If scripts were simply a list of commands to be executed in order, their usefulness would be severely limited. The flow of control statements let scripts loop and branch. It is considered good practice to keep flow of control statements on separate lines, but the script syntax does not require this. There are two branching statements, `if...endif` and `docase...endcase`, and three looping statements, `repeat...until`, `while...wend` and `for...next`. You can also use user-defined functions and procedures with the `Func` and `Proc` statements.

if...endif

The `if` statement can be used in two ways. When used without an `else`, a single section of code can be executed conditionally. When used with an `else`, one of two sections of code is executed. If you need more than two alternative branches, the `docase` statement is usually more compact than nesting many `if` statements.

```
if expression then           'The simple form of an if
    zero or more statements;
endif;
if expression then         'Using an else
    zero or more statements;
else
    zero or more statements;
endif;
```

If *expression* is non-zero, the statements after the `then` are executed. If *expression* is zero, only the statements after the `else` are executed. The following code adds 1 or 2 to a number, depending on it being odd or even:

```
if num% mod 2 then
    num%:=num%+2;           'note that the semicolons before...
else                        '...the else and endif are optional.
    num%:=num%+1;
endif;

'The following are equivalent
if num% mod 2 then num%:=num%+2 else num%:=num%+1 endif;
num% += num% mod 2 ? 2 : 1; 'An alternative using ?:
```

docase...endcase

These keywords enclose a list of `case` statements forming a multi-way branch. Each `case` is scanned until one is found with a non-zero expression, or the `else` is found. If the `else` is omitted, control passes to the statement after the `endcase` if no `case` expression is non-zero. Only the first non-zero `case` is executed (or the `else` if no `case` is non-zero).

```
docase
  case exp1 then
    statement list;
  case exp2 then
    statement list;
  ...
  else
    statement list;
endcase;
```

This example displays the type of a file handle:

```
'case.s2s
var i%,m$;
i% := ViewKind();           'get type of the current view
docase
  case i% = 0 then m$ := "time";
  case i% = 1 then m$ := "text";
  case i% = 2 then m$ := "output sequence";
  case i% = 3 then m$ := "script";
  case i% = 4 then m$ := "result";
  case i% = 8 then m$ := "external text";
  case i% = 9 then m$ := "external binary";
  case i% = 10 then m$ := "Spike2";
  case i% = 12 then m$ := "XY view";
  else m$ := "something else...";
endcase;
message("Current window is of type "+m$);
```

The following example sets a string value depending on the value of a number:

```
var base%:=8,msg$;
```

```
docase
  case base%=2 then msg$ := "Binary";
  case base%=8 then msg$ := "Octal";
  case base%=10 then msg$ := "Decimal";
  case base%=16 then msg$ := "Hexadecimal";
  else msg$ := "Pardon?";
endcase;
```

repeat...until

The statements between `repeat` and `until` are repeated until the expression after the `until` keyword evaluates to non-zero. The body of a repeat loop is always executed at least once. If you need the possibility of zero executions, use a `while` loop. The syntax of the statement is:

```
repeat
  zero or more statements;
until expression;
```

For example, the following code prints the times of all data items on channel 3 (plus an extra -1 at the end):

```
var time := -1;           'start time of search
repeat
  time := NextTime(3, time); 'find next item time
  PrintLog("%f\n", time);    'display the time to the log
until time<0;             'until no data found
```

You can break out of a repeat loop early using the `break` statement. You can skip to the `until` statement with the `continue` statement.

```
var t;
repeat
  t := NextTime(3, t);      'get a time
  if (t<0) then break endif; 'stop if time not found
  if (t<10) then continue endif; 'skip if not enough time yet
  PrintLog("At time %5.2f 10 second count = %d\n", t, Count(3, t-10, t));
until t >= MaxTime(3);
```

while...wend

The statements between the keywords are repeated while an expression is not zero. If the expression is zero the first time, the statements in between are not executed. The `while` loop can be executed zero times, unlike the `repeat` loop, which is always executed at least once.

```
while expression do
  zero or more statements;
wend;
```

The following code fragment, finds the first power of two that is greater than or equal to some number:

```
var test%:=437, try%:=1;
while try%<test% do      'if try% is too small...
  try% := try% * 2;      '...double it
wend;
```

You can break out of a while loop early using the `break` statement. You can skip back to the start of the loop with the `continue` statement.

```
var i% := 0;
while i% <= 999 do
  ...
  if StopEarly() then
    break
  else
    if SkipRestOfLoop() then continue endif;
  endif;
  ...
wend;
```


for...next

A `for` loop executes a group of statements a number of times with a variable changed by a fixed amount on each iteration. The loop can be executed zero times. The syntax is:

```
for v := exp1 to exp2 {step exp3} do
    zero or more statements;
next;
```

- v This is the loop variable and may be real or integer. It must be a simple variable, not an array element.
- exp1 This expression sets the initial variable value before the looping begins.
- exp2 This expression is evaluated once, before the loop starts, and is used to test for the end of the loop. If `step` is positive or omitted (when it is 1), the loop stops when the variable is greater than `exp2`. If `step` is negative, the loop stops when the variable is less than `exp2`.
- exp3 This expression is evaluated once, before the loop starts, and sets the increment added to the variable when the `next` statement is reached. If there is no `step exp3` in the code, the increment is 1. The value of `exp3` can be positive or negative.

The following example prints the squares of all the integers between 10 and 1:

```
var num%;
for num% := 10 to 1 step -1 do
    PrintLog("%d squared is %d\n", num%, num% * num%);
next;
```

This is a more complicated example that prints prime numbers:

```
const MaxN% := 1000;      'the maximum number to search
var prime%[MaxN%],i%,t%; 'the "Sieve" and loop variables
for i%:=1 to MaxN%-1 do  'ignore 0 as not included in search
    prime%[i%] := 1      'mark all numbers as possible primes
next;
for t%:=2 to MaxN%-1 do  'start search at 2 as 1 is special case
    if prime%[t%] then  'if number remains, must be prime
        for i% := 2*t% to MaxN%-1 step t% do
            prime%[i%]:=0 'remove all multiples of prime number
        next;
    endif;
next;
t% := 0;                  'use this to put output in columns
for i% := 1 to MaxN%-1 do 'search the "sieve" for next prime
    if prime%[i%] then   'have we found one?
        if t% mod 10 = 0 then PrintLog("\n") endif;
        t% := t% + 1;    'count the number
        PrintLog("%5d", i%); 'output the prime
    endif;
next;
```

If you want a `for` loop where the end value and/or the `step` size are evaluated each time round the loop you should use a `while...wend` or `repeat...until` construction. You can break out of a `for` loop early using the `break` statement. You can skip to the next statement with the `continue` statement.

break, continue

Within the three looping statements (`for...next`, `repeat...until` and `while...do`) you can use the `break` and `continue` statements. The `break` statement jumps out of the enclosing looping statement; the `continue` statement jumps to the evaluation of the expression that determines if a `repeat` or `while` will run again and to the next in a `for` loop. For example:

```
var i%;
for i% := 0 to 1000 do
    if StopEarly() then break endif;
    if SkipToNext() then continue endif;
    DoSomething(i%);
next;                               'continue jumps here
```

```
DoSomethingElse(); 'break jumps to this statement
```

It is an error to use either of these statements outside a loop. If you need to jump out of more than one level of looping statements, put the code in a `Proc` or `Func` and use the `return` statement to jump out.

The `break` and `continue` statements are useful because they give you a way to jump out of deeply nested `if` or `docase` statements inside a loop in a similar way that the `return` statement allows you to jump out of the middle of a `Proc` or `Func`.

The `break` and `continue` statements were added at Spike2 version 7; if you use them your script will not compile in older versions of Spike2.

halt

The `Halt` keyword ends a script. A script also terminates if the path of execution reaches the end of the script. When a script halts, any open external files associated with the `Read()` or `Print()` functions are closed and any windows with invalid regions are updated. Control then returns to the user unless `ScriptRun()` has been used to set the next script to run. The `Halt` statement can occur at the outer level of a script or in a function or procedure. For example:

```
Proc DoSomething(arg)
if arg < 0 then
    Message("Bad argument to DoSomething(%g)", arg);
    halt;
endif;
...
end;
```

Functions and procedures

A user-defined function is a named block of code. It can access global variables and create its own local variables. Information is passed into user-defined functions through arguments. Information can be returned by giving the function a value, by altering the values of arguments passed by reference or by changing global variables.

User-defined functions that return a value are introduced by the `func` keyword, those that do not are introduced by the `proc` keyword. The `end` keyword marks the end of a function. The `return` keyword returns from a function. If `return` is omitted, the function returns to the caller when it reaches the `end` statement. Arguments can be passed to functions by enclosing them in brackets after the function. Functions that return a value or a string have names that specify the type of the returned value. A function is defined as:

```
func name({argument list})      or   proc name({argument list})
{var local-variable-list;}      {var local-variable-list;}
statements including return x;   statements including return;
end;                             end;
```

There is no semicolon at the end of the argument list because the argument list is not the end of the `func` or `proc` statement; the `end` keyword terminates the statement. Functions may not be nested within each other.

This example function finds the greatest common divisor of two integers:

```
var n1%, n2%;
n1% := Input("First number", 10000, 1, 2000000000);
n2% := Input("Second number", 30000, 1, 2000000000);
Message("Greatest common divisor of %d and %d is %d", n1%, n2%, gcd%(n1%, n2%));
halt;
func gcd%(m%, n%) 'm% and n% both > 0!
var d%;
while n% do
    d% := Max(m%, n%) mod Min(m%, n%);
    m% := Min(m%, n%); 'get smaller value
    n% := d%
wend;
return m%;
end;
```

return

The `return` keyword is used in a user-defined function to return control to the calling context. In a `proc`, the `return` must not be followed by a value. In a `func`, the `return` should be followed by a value of the correct type to match the function name. If no return value is specified, a `func` that returns a real or integer value returns 0, and a `func` that returns a string value returns a string of zero length.

You can use a `return` statement in a function or procedure to break out of a loop or an `if` or `case` statement. If you are using a `return` to break out of a loop early and do not need to be compatible with version 6, you could consider using the `break` statement.

Call tips

You can define pop-up call tips that appear when you hover the mouse over a user-defined `Proc` or `Func` name. These are enabled in the script file settings dialog.

Argument lists

The argument list is a list of up to 20 variable names separated by commas. There are two ways to pass arguments to a function: by value and by reference:

- | | |
|-----------|---|
| Value | Arguments passed by value are local variables in the function. Their initial values are passed from the calling context. Changes made in the function to a variable passed by value do not affect the calling context. |
| Reference | Arguments passed by reference are the same variables (by a different name) as the variables passed from the calling context. Changes made to arguments passed by reference do affect the calling context. Because of this, reference arguments must be passed as variables (not expressions or constants) and the variable must match the type of the argument (except we allow you to pass a real variable where an integer variable is expected). |

Simple (non-array) variables are passed by value. Simple variables can be passed by reference by placing the `&` character before the name in the argument list declaration.

Arrays and sub-arrays are always passed by reference. Array arguments have empty square brackets in the function declaration, for example `one[]` for a vector and `two[][]` for a matrix. The number of dimensions of the passed array must match the declaration. The array passed in sets the size of each dimension. You can find the size of an array with the `Len()` function. An individual array element is treated as a simple variable.

If you use the `trans()` or `diag()` operators to pass the transpose or diagonal of an array to a function, the array is still passed by reference and changes made in the function to array elements will change the corresponding elements in the original data.

Default arguments

Simple (non array) arguments that are passed by value can also specify a default value. This is done by following the variable name by `:= value`, for example:

```
func fred(a := 1, b$ := "", c% := 0)
...
return a;
end;
```

When arguments are defined with default value you can omit them entirely when they are trailing arguments, or skip over them if they are not. For example, all the following are legitimate:

'call as	equivalent to
<code>fred();</code>	<code>'fred(1,"",0)</code>
<code>fred(2);</code>	<code>'fred(2,"",0)</code>
<code>fred(,"x");</code>	<code>'fred(1,"x",0)</code>
<code>fred(,,6);</code>	<code>'fred(1,"",6)</code>

Default arguments are just a convenience. The code is generated exactly as if you had supplied the arguments. A common use is to extend an established command to cover a special case. By adding an addition argument with a default value you can leave all your existing code unchanged and then use the extra argument only when required.

Examples of user-defined functions

```
proc PrintInfo()           'no return value, no arguments
PrintLog(ChanTitle$(1));  'Show the channel title
PrintLog(ChanComment$(1)); 'and the comment
return;                   'return is optional in this case as...
end;                       '...end forces a return for a proc

func sumsq(a, b)           'sum the square of the arguments
return a*a + b*b;
end;

func removeExt$(name$)    'remove text after last . in a string
var n := 0, k := 1;
  repeat
    k:=InStr(name$,".",k); 'find position of next dot
    if (k > 0) then        'if found a new dot...
      n := k;              '...remember where
    endif
  until k=0;               'until all found
if n=0 then
  return name$;           'no extension
else
  return Left$(name$,n-1);
end;

proc sumdiff(&arg1, &arg2) 'returns sum and difference of args
arg1 := arg1 + arg2;      'sum of arguments
arg2 := arg1 - 2*arg2;    'original arg1-arg2
return;                   'results returned via arguments
end;

func sumArr(data[])       'sum all elements of a vector
var sum:=0.0;             'initialise the total
var i%;                  'index
for i%:=0 to Len(data[])-1 do
  sum := sum + data[i%];  'of course, ArrSum() is much faster!
next;
return sum;
end;

Func SumArr2(data[][]])  'Sum of all matrix elements
var rows%,cols%,i%,sum;  'sizes, index and sum, all set to 0
rows% := Len(data[0][])  'get sizes of dimensions...
cols% := Len(data[][0]);  '...so we can see which is bigger
if rows%>cols% then      'choose more efficient method
  for i%:=0 to cols%-1 do sum += ArrSum(data[i%][]) next;
else
  for i%:=0 to rows%-1 do sum += ArrSum(data[][i%]) next;
endif;
return sum;
end;
```

Variables declared within a function exist only within the body of the function. They cannot be used from elsewhere. You can use the same name for variables in different functions. Each variable is separate. In addition, if you call a function recursively (that is it calls itself), each time you enter the function, you have a fresh set of variables.

Scope of user-defined functions

Unlike global variables, which are only visible from the point in the script in which they are declared onwards, and local variables, which are visible within a user-defined function only, user-defined functions are visible from all points in the script.

Mutually recursive functions

Because functions are globally visible, you may define two functions that call each other (mutually recursive). If you do this it is your responsibility to ensure that there is a way out of the mutual recursion. If you do not, the script will compile, but will crash at run time as it will (eventually) exhaust system memory.

Functions as arguments

The script language allows a function or procedure to be passed as an argument. The function declaration includes the declaration of the function type to be passed. Functions and procedures can occur before or after the line in which they are used as an argument.

```
proc Sam(a,b$,c%)
...
end;

func Calc(va)
return 3*va*va-2.0*va;
end;

func PassFunc(x, func ff(realarg))
return ff(x);
end;

func PassProc(proc jim(realarg, strArg$:= "", son%:=0))
jim(1.0, "hello", 3);
jim(2.0);           'using default arguments
end;

var val := PassFunc(1.0, Calc); 'pass function
PassProc( Sam );           'pass procedure
```

The declaration of the procedure or function argument is exactly the same as for declaring a user-defined function or procedure. This means you can also define default arguments. When passing the function or procedure as an argument, just give the name of the function or procedure; no brackets or arguments are required. The compiler checks that the argument types of a function passed as an argument match those declared in the function header. See the `ToolBarSet()` function for an example.

Although user-defined functions and built-in functions are very similar, you are not allowed to pass a built-in function as an argument to a built-in function. Further, you cannot pass a built-in function to a user-defined function if it has one or more arguments that can be of more than one type. For example, the built-in `Sin()` function can accept a real argument, or a real array argument, and so cannot be passed. If you need to pass such a function you must wrap it in a user-defined function:

```
func USin(x) return Sin(x) end; 'USin(x) has an unambiguous signature
```

Channel specifiers

Several built-in script commands use a channel specifier to define a list of 1 or more channels. This argument is always called `cSpC` in the documentation. This argument stands for a string, an integer array or an integer.

- `cSpC$` This holds a list of channel numbers and channel ranges, separated by commas. A channel range is a start channel number followed by an end channel number separated by two dots. The end channel number can be less than the start channel number. For example "1,3,5..7,12..9" is a list of channels 1, 3, 5, 6, 7, 9, 10, 11 and 12. If a `cSpC$` contains an error, the error terminates parsing of the string, but any channels found before the error are preserved. Use `ChanList()` to convert a `cSpC$` to a list of channel numbers.
- `cSpC%[]` The first element of the vector is the number of channels. The remaining elements are channel numbers. It is an error for the vector passed in to be shorter than the number of channels + 1.
- `cSpC%` This is either a channel number, or -1 for all channels, -2 for visible channels or -3 for selected channels.

In a result view, channel numbers start at 1, but we accept 0 (meaning 1) to avoid breaking scripts written for older versions of Spike2. Some commands expect channels of specific types; channels that do not meet the type requirements are removed from the list. It is usually an error for a channel specification to generate an empty list.

Include files

There are times when you will want to reuse definitions or user-defined procedures and functions in multiple projects. You can do this by pasting the text into your script, but it can be more convenient to use the `#include` command to include script files into a script. A file that is included can also include further files. We call these *nested* include files. Only the first `#include` of a file has any effect. Subsequent `#include` commands that refer to the same file are ignored. This prevents problems with script files that include each other and stops multiple definitions when two files include a common file. A `#include` command must be the first non-white space item on a line. There are two forms of the command:

```
#include "filename" 'optional comment
#include <filename> 'optional comment
```

where `filename` is either an absolute path name (starting with a `\` or `/` or containing a `:`), for example `C:\Scripts\MyInclude.s2s`, or is a relative path name, for example `include.s2s`. The difference between the two command forms lies in how relative path names are treated. The search order for the first form starts at item 1 in the following list. The search for the second form starts at item 3.

1. The folder where the file with the `#include` command lives. If this fails...
2. The folder of the file that included that file up to the top of the list of nested include files. If this fails...
3. Any `\include` folder in the `Spike8` folder inside your `My Documents` folder. If this fails...
4. Any `\include` folder in `\Users\Public\Public Documents\Spike8Shared`. If this fails...
5. Any `\include` folder in the folder in which `Spike2` is installed. If this fails...
6. Search the current folder.

Included files are always read from disk, even if they are already open. If you have set the `Edit` menu Preferences option to save modified scripts before running, modified include files are automatically saved when you compile. If this option is not set, the script compiler will stop the compilation with an error if it finds a modified include file. You must save the included file to compile your script.

There are no restrictions on what can be in an included file. However, they will normally contain constant and variable definitions and user-defined procedures and functions. It is usually a good idea to have all your `#include` commands at the start of a script so that anyone reading the source is aware of the included files.

The `#include` command was added to `Spike2` at version 6.03 and is not recognised by any version before this. A typical file using `#include` might start with:

```
'$Example|Example of use if include files
#include <sysinc.s2s> 'my system specific includes
#include "include\proginc.s2s" 'search script relative folder
var myvar; 'start of my code...
```

Opening included files

If you right-click on a line that holds an include command, and `Spike2` can locate the included file, the context menu will hold an entry to open the file. The search for the file follows that described above, except that it omits step 2.

You can also open an include file by right-clicking on the name of any user-defined `Proc` or `Func` that is defined in an included file and selecting the `Go to` command. This will open the include file with the text caret at the start of the `Proc` or `Func`.

Include files and debugging

You can debug a script that uses included files in exactly the same way as one that does not. If you step into a user-defined function or procedure that is in an included file, the included file will open and the stepping marker will show where you have reached. If you want to set a break point in an included file before you run the script, open the included file, set the break point and then run the script (leaving the included file open).

Included files that are open within `Spike2` are hidden from the script `ViewList()` command in the same way that the running script is not visible. However, unlike the running script, whose view handle can be obtained with the `App()` command, there is currently no mechanism built into the system to let you find the handles of included files.

Program size limits

The use of `#include` makes it much easier to construct very large scripts. However, there are some limits on the size of a script that you should bear in mind before trying to amalgamate every script you have ever written into one "superscript":

- The maximum instructions in a compiled script is currently set at 1000000. The number of instructions your script uses is displayed when you compile a script.
- A script file can contain up to 1000000 lines and you can have up to 4095 included files. However, you will almost certainly run out of instruction space before you hit these limits.
- Every variable, constant, procedure and function you declare generates a script object. The maximum number of objects is currently around 32767.
- The maximum number of global objects (variables and constants that are visible throughout the script) is 10000 (was 15000 before version 6.10). Note that there are about 500 global objects defined before any you create with a script, so you only have space for some 9500.
- The maximum number of local variables in all `proc` and `func` items is 12000 (was 7000 before version 6.10).
- The maximum number of arguments to a `proc` or `func` is 20.
- The maximum number of strings that are longer than a few characters is 7000.
- The maximum number of `proc` and `func` items is 2767.
- The maximum number of characters in distinct literal strings is around 1000000. Before version 6.04 the limit was around 65000 characters.
- The maximum size of an array (product of all the array dimensions) is 100,000,000 elements.

We can expand these limits if anyone manages to hit them.

Script functions by topic

This section lists script commands by function.

Windows and views

These commands open, close, manipulate, position, display, size, colour and create windows (views). These commands apply to windows in general. See the sections on Time, Result, XY and Text windows for more specific commands.

App	Get the application view handle and version and special views
ChanNumbers	Hide and show channel numbers
Colour	Get or set the palette entry associated with a screen item
Draw	Draw invalid regions of the view (and set x axis range)
DrawAll	Update all invalid regions in all views
EditCopy	Copy window to clipboard
FileClose	Closes a window or windows
FileComment\$	Gets and sets the file comment for time views
FileConvert\$	Convert a foreign file to a Spike2 file and open it
FileName\$	Gets the file name associated with a window
FileNew	Opens an output file or a new text or data window
FileOpen	Opens an existing file (in a window)
FilePrint	Prints a range of data from the current view
FilePrintVisible	Prints the current view
FilePrintScreen	Prints all text-based, time and result views
FileQuit	Closes the application
FileSave	Save a view
FileSaveAs	Save a view in variety of formats
FontGet	Read back information about the font
FontSet	Set the font for the current window
FocusHandle	Get handle of script-controllable window with the focus
FrontView	Get or set the front window on screen
Grid	Get or set the visibility of the axis grid
LogHandle	Gets the view handle of the log window
Modified	Get and set the modified and read-only state of views
PaletteGet	Get the RGB colour of a palette entry
PaletteSet	Set the RGB colour of a palette entry
View	Change or override current view and get view handle
ViewColour	Override application colours for a view using the palette
ViewColourGet	Get view item colours as RGB
ViewColourSet	Override application colours for a view as RGB
ViewFind	Get a view handle from a view title
ViewKind	Get the type of the current view
ViewList	Form a list of handles to views that meet a specification
ViewStandard	Returns a window to a standard state
ViewUseColour	Get and set monochrome/colour use for view
Window	Sets the window size and position
WindowGetPos	Get window position
WindowSize	Changes the window size
WindowTitle\$	Gets or changes the window title
WindowVisible	Sets or gets the visibility of the window (hide/show)
XAxis	Get or set visibility of the x axis
XAxisMode	Control how the x axis is drawn
XAxisStyle	Control x axis tick spacing and time view hh:mm:ss mode
XScroller	Get or set visibility of x axis scroll bar
XUnits\$	Get units of the x axis
YAxis	Get or set the visibility of the y axis
YAxisMode	Control how the y axis is drawn
YAxisStyle	Control y axis tick spacing

Channel commands

These commands manipulate data channels.

BinError	Get or set error bar information
Binsize	The sample interval for waveforms, otherwise time resolution
BurstMake	Extract bursts to a memory channel from an event channel
BurstRevise	Modify a list of bursts in a memory channel
BurstStats	Collect statistics on bursts in preparation for <code>BurstRevise()</code>
Chan\$	Get the symbolic name for a channel
ChanCalibrate	Calibrate waveform or WaveMark channels from known values
ChanColour	Override drawing mode based channel colours by palette
ChanColourGet	Get channel colour as RGB
ChanColourSet	Set channel colour override as RGB
ChanComment\$	Get or set the channel comment
ChanData	Fill an array with a waveform or event times
ChanDelete	Delete a channel from a time view
ChanDuplicate	Duplicate channels in a time view
ChanHide	Make a channel invisible
ChanImage	Set a bitmap to appear behind a channel
ChanIndex	Select which RealMark item to display and measure
ChanKey	Control sonogram and XY view keys
ChanKind	Get the type of a channel
ChanList	Get a list of channels meeting a specification
ChanMeasure	Make the same measurements as the Cursor Regions dialog
ChanNew	Create a new channel for <code>ChanWriteWave()</code> to write to
ChanOffset	Set waveform and WaveMark value for input of zero
ChanOrder	Modify the channel order and y axis grouping
ChanPenWidth	Set the pen width for a time or result view channel
ChanProcessAdd	Add new processing option to a waveform or RealWave channel
ChanProcessArg	Read or modify an argument of a channel process
ChanProcessClear	Delete a process, all processes for a channel, or all processes
ChanProcessCopy	Copy channel processes (rectify, smooth) to other channels
ChanProcessInfo	Get information about the processes attached to a channel
ChanPort	Get the physical sampling port of a time view data channel
ChanScale	Set waveform and WaveMark channel scale factor
ChanSave	Copy time view channels between views with optional time shift
ChanSearch	Search a channel for a feature (peak, level crossing, slope...)
ChanSelect	Select and report on selected state of channels
ChanShow	Make a channel visible
ChanTitle\$	Get or set the channel title string
ChanUnits\$	Get or set the channel units
ChanValue	Get channel data at a particular time or x axis position
ChanVisible	Get the visibility state of a channel
ChanWeight	Change the relative vertical space of a channel
ChanWriteWave	Write data to a waveform or RealWave channel
Count	Count items in a time range, sum bins in result view
DrawMode	Get or set display mode for a channel
DupChan	Get information about duplicate channels
EventToWaveform	Convert an event channel into a waveform channel
FitLine	Fit a straight line to waveform or result channel
LastTime	Find the previous item in a channel (and return values)
MarkEdit	Edit marker codes and other information
MarkInfo	Get information on extended marker types
MarkMask	Set the marker filter for a channel
MarkSet	Set the marker codes for data in a time range
Maxtime	Time of last item on the channel
MemChan	Create a channel in memory
MemDeleteItem	Delete one or more items from a memory channel
MemDeleteTime	Delete one or more items based on time in a memory channel
MemGetItem	Get information on item in memory channel

MemImport	Import items into a memory channel
MemSave	Write the contents of a memory channel to the data file
MemSetItem	Edit or add an item in a memory channel
Minmax	Find minimum and maximum values (and positions)
NextTime	Find the next item in a channel (and return values)
VirtualChan	Create data channels from existing channels and math functions

Time views

These commands manipulate time views.

BinToX	Convert time units to x axis units
DrawMode	Get or set display mode for a channel
Dup	Get the view handle of duplicates of the current view
EventToWaveform	Create a waveform channel from events
ExportChanFormat	Set channel format for export
ExportChanList	Set list of channels for export
ExportRectFormat	Set rectangular time view export
ExportTextFormat	Set format for text output of channels
FileApplyResource	Apply a resource file to the current time view
FileGlobalResource	Set a global resource file
FileSaveResource	Create a resource file matching the current time view
FileTimeBase	Get and optionally set the underlying time units
Maxtime	Maximum x axis value for any channel in the view
Optimise	Set reasonable y range for channels with axes
ReRun	Get or set the rerunning state of a channel
VerticalMark	Vertical marker display control
ViewOverdraw	Add a list of overdraw trigger times to a triggered display
ViewOverdraw3D	Set the drawing parameters for overdrawn data in 3D mode
ViewTrigger	Control on-line triggered displays and off-line display stepping
VirtualChan	Create data channels from existing channels and math functions
WindowDuplicate	Duplicate a time view
XLow	Time of the start of the displayed area in seconds
XHigh	Time of the end of the displayed area in seconds
XRange	Set the x axis range for next draw
XScroller	Show or hide the x axis scroll bar and controls
XTitle\$	Get x axis title
XToBin	Convert x axis units to time units
YAxisLock	Get and set the locking state of grouped channels
YRange	Set y axis range for a channel
YHigh	Get upper limit of y axis for a channel
YLow	Get lower limit of y axis for a channel

Result views

These commands manipulate result views. Result views can also be treated as arrays; so all the array arithmetic commands are available.

BinError	Get or set error bar and bin count information
Binsize	Width of each bin in the x direction
BinToX	Convert bin number to x axis value
DrawMode	Get or set display mode for a channel
Maxtime	Number of bins in the result view
Minmax	Find minimum and maximum values (and positions)
Optimise	Set reasonable y range for display
RasterAux	Deprecated, use RasterSort () and RasterSymbol ()
RasterGet	Read back raster information
RasterSet	Set raster information
RasterSort	Get and set the sorting values
RasterSymbol	Get and set the time values for raster symbols
Sweeps	Gets and sets the number of items added to result view

XHigh	Bin number of the end of the displayed area
XLow	Bin number of the start of the displayed area
XRange	Set the x axis range for next draw
XScroller	Show or hide the x axis scroll bar and controls
XTitle\$	Get and set x axis title
XToBin	Convert x axis value to bin number
YAxisLock	Get and set the locking state of grouped channels
YHigh	Get upper limit of y axis
YLow	Get lower limit of y axis
YRange	Set y axis range

XY views

XY views hold from 1 to 256 channels of (x, y) co-ordinate pairs that can be displayed in a variety of styles. Most functions that work for views in general will work for an XY view. The following are the XY view specific commands.

ChanKey	Control keys in XY views and sonograms
MeasureToXY	Create an XY view and associated measurement process
XYAddData	Add data points to a channel of an XY view
XYColour	Set the colour of a channel
XYCount	Return the number of XY data points in a channel
XYDelete	Delete one or more data points from a channel
XYDrawMode	Control how a channel is drawn
XYGetData	Read data back from an XY channel
XYInChan	Count and get points in a shape defined by a channel
XYInCircle	Count the number of XY points within a circle
XYInRect	Count the number of XY points inside a rectangle
XYJoin	Get or set the point joining method
XYKey	Control the display of the XY view channel key
XYRange	Get rectangle containing one or more channels
XYSetChan	Create or modify an XY channel
XYSize	Get or set maximum size of an XY channel
XYSort	Change the sort (and draw) order of a channel

Text views

Text views (which includes Script views and output sequencer views) hold text organised in lines.

EditClear	Clear selected text
EditCopy	Copy selected text to the clipboard
EditCut	Cut selected text to the clipboard
EditFind	Search for text
EditPaste	Replace selection with clipboard contents
EditReplace	Replace matching selection and search for next
EditSelectAll	Select all text
FileName\$	Get associated file name
FilePrint	Print current file
FilePrintVisible	Print selection or line holding text caret
FileSave	Save as current name
FileSaveAs	Save with a different name
FontGet	Get the current font characteristics
FontSet	Set the current font characteristics
Gutter	Control display of the gutter
LogHandle	Get handle of the log view
Modified	Get and set the modified and read-only state of views
MoveBy	Move the text caret relative to the current position
MoveTo	Move the text caret to an absolute position
Print	Print text at the text caret position
PrintLog	Print text to the end of the log view

Read	Read the text line containing the caret
ReadSetup	Setup how read interprets item separators
TabSettings	Get and set the Tab settings
ViewLineNumbers	Control display of line numbers
ViewMaxLines	Get and set the maximum lines in a view
ViewZoom	Get and set the zoom factor
XHigh	Get line of first visible line plus visible line count
XLow	get the first visible line in the view

Vertical cursors

The following commands control the vertical cursors.

Cursor	Set or get the position of a cursor
CursorActive	Set or get the active cursor mode
CursorActiveGet	Read back active cursor parameters
CursorDelete	Delete a designated cursor
CursorExists	Test if a cursor exists
CursorLabel	Set or get the cursor label style
CursorLabelPos	Set or get the cursor label position
CursorNew	Add a new cursor (at a given position)
CursorOpen	Open the value and regions dialogs, test if open
CursorRenumber	Renumber the cursors in ascending position order
CursorSearch	Trigger active cursor searches from the script
CursorSet	Set the number (and position) of vertical cursors
CursorValid	Test if an active cursor search succeeded
CursorVisible	Get or set cursor visibility

Horizontal cursors

Horizontal cursors are controlled from a script with the following commands.

HCursor	Set or get the position of a horizontal cursor
HCursorChan	Gets the channel that a horizontal cursor belongs to
HCursorDelete	Delete a designated horizontal cursor
HCursorExists	Test if a cursor exists
HCursorLabel	Gets or sets the horizontal cursor style
HCursorLabelPos	Gets or sets the horizontal cursor position
HCursorNew	Add a new horizontal cursor on a channel (at a given position)
HCursorRenumber	Renumsbers the cursors from bottom to the top of the view

Editing operations

These functions mimic the Edit menu commands and provide additional functionality.

EditClear	Delete text from a text window at the caret
EditCopy	Copy the current selection to the clipboard
EditCut	Delete the current selection to the clipboard
EditFind	Find text
EditPaste	Paste the clipboard into the current text field
EditReplace	Find and replace text
EditSelectAll	Select the entire text or cursor window contents
MoveBy	Move relative to current position
MoveTo	Move to a particular place
Selection\$	This function returns the text that is currently selected

Text files

Spike2 can create, read and write text files. You can also open a text file into a window.

FileNew	Open a new text file in a window
FileOpen	Open a text file in a window or for reading and writing
FileSaveAs	Save a view in variety of formats, including text
Print	Write formatted output to a file or log window
Read	Extract data from a text file
ReadSetup	Set separators and delimiters for Read() and ReadStr()
TabSettings	Get or set the tab and indent settings for a text view

Binary files

Spike2 can read and write binary files. Binary files provide links to other programs and are generally more efficient than text for transferring large quantities of data.

FileClose	Close a file opened in binary mode
FileOpen	Open an external file in binary mode
BRead	Extract 32-bit integer, 64-bit real and string data from a file
BReadSize	Extract 8 and 16-bit integer and 32-bit real data from a file
BRWEndian	Change byte order for read/write of numbers in binary files
BSeek	Change the current file position for next read or write
BWrite	Write 32-bit integer and 64-bit real data to a file
BWriteSize	Write 8 and 16-bit integer, 32-bit real and string data to a file

File system

Spike2 can read file information, change the current directory, delete and copy files and convert foreign files into Spike2 format. You can also execute external files.

FileConvert\$	Convert a foreign file to a Spike2 file and open it
FileCopy	Copy one file to a new location
FileDate\$	Retrieve date of creation of Spike2 data file
FileDelete	Delete one or more files
FileList	Get a list of files or directories
FilePath\$	Get the current directory or directory for new data files
FilePathSet	Change the current directory or directory for new data files
FileTime\$	Retrieve time of creation of Spike2 data file
FileTimeDate	Retrieve time and date of creation of Spike2 data file as numbers
FileTimeDateSet	Set the time of creation (start of sampling) of a data file

User interaction commands

These commands exchange information with the user or let the user interact with the data.

Inkey	Return key code of last key user pressed
Input	Prompt user for a number in a defined range
Input\$	Prompt user for a string with a list of acceptable characters
Interact	Allow user to interact with data
Keypress	Detect if the Inkey() function would read a character
Message	Display a message in a box, wait for OK
Print	Formatted text output to a file or window
PrintLog	Formatted text output to the Log window
Print\$	Formatted text output to a string
Query	Ask a user a question, wait for response
Sound	Play a tone or a .wav file
Speak	Convert text to speech on systems that support this
Yield	Give idle time to the system; delay for a time

YieldSystem Surrender current time slice and sleep Spike2

You can build simple dialogs, with a set of fields stacked vertically or you can build free-format dialogs (but with more work to define the positions of all the dialog fields):

DlgAllow	Set allowed actions and change and idle functions
DlgButton	Add buttons to the dialog and modify existing buttons
DlgChan	Define a dialog entry as prompt and channel selection
DlgCheck	Define a dialog item as a check box
DlgCreate	Start a dialog definition
DlgEnable	Enable and disable dialog items in change or idle functions
DlgGetPos	Get the dialog position in change and idle functions
DlgGroup	Position a group box in the dialog
DlgInteger	Define a dialog entry as prompt and integer number input
DlgLabel	Define a dialog entry as prompt only
DlgList	Define a dialog entry as prompt and selection from a list
DlgMouse	Set the mouse pointer position when the dialog opens
DlgReal	Define a dialog entry as prompt and real number input
DlgShow	Display the dialog, get values of fields
DlgSlider	Define a dialog entry as a slider for integer number input
DlgString	Define a dialog entry as prompt and string input
DlgText	Define a fixed text string for the dialog
DlgValue	Gives access to dialog fields in change and idle functions
DlgVisible	Show and hide dialog items in change or idle functions
DlgXValue	Define a dialog entry to collect an x axis value

These commands control the various toolbars and link script functions to the toolbar.

App	Get the view handle of the toolbars
MousePointer	Add extra mouse pointers for use with the toolbar
Toolbar	Let the user interact with the toolbar
ToolbarClear	Remove all defined buttons from the toolbar
ToolbarEnable	Get or set the enables state of toolbar buttons
ToolbarMouse	Link mouse actions to user functions
ToolbarSet	Add a button (and associate a function with it)
ToolbarText	Display a message using the toolbar
ToolbarVisible	Get or set the visibility of the toolbar
SampleBar	Controls the sample toolbar buttons
ScriptBar	Controls the script toolbar buttons

Analysis

These functions create new result views and analyse data into the result views.

BinError	Get or set error bar information
MeasureToXY	Create an XY view and associated measurement process
MeasureX	Set the x part of a measurement
MeasureY	Set the y part of a measurement
RasterAux	Set additional data for sorted rasters (deprecated)
RasterGet	Read back raster information
RasterSet	Set raster information
RasterSort	Get and set the sorting values
RasterSymbol	Get and set the time values for raster symbols
SetEvtCrl	Create result view for event correlation
SetEvtCrlShift	Set shift for shuffled event correlation
SetINTH	Create result view for interval histogram
SetPhase	Create result view for phase histogram
SetPower	Create result view for power spectrum
SetPSTH	Create result view for post/peri-stimulus histogram
SetWaveCrl	Create result view for waveform correlation
SetWaveCrlDC	Set DC use/ignore for SetWaveCrl ()
SetAverage	Create result view for waveform copy/average/sum
SetResult	Create result view for user-defined data

Sweeps	Number of items added to result view
Process	Process data into result view
ProcessAll	Process all result views attached to a time view
ProcessAuto	Process data automatically during sampling
ProcessTriggered	Process triggered by data during sampling

Mathematical functions

The following mathematical functions are built into Spike2. You can apply many of these functions to real arrays by passing an array to the function.

Abs	Absolute value of a number or array
ATan	Arc tangent of number or array
BinomialC	Number of ways of choosing k from n (binomial coefficient)
Ceil	Ceiling of a number or array (next highest integral value)
Cos	Cosine of a number or array
Cosh	Hyperbolic cosine of a number or array
Exp	Exponential function of a number or array
Floor	Floor of a number or array (next lowest integral value)
Frac	Remove integral part of a number or array
GammaP	Incomplete Gamma function, used to calculate probabilities
Ln	Natural logarithm of a number or array
LnGamma	Natural logarithm of the Gamma function (use for big factorials)
Log	Logarithm to base 10 of a number or array
Max	Finds maximum of array or variables
Min	Finds minimum of array or variables
Pow	Raise a number or an array to a power
Rand	Generate pseudo-random numbers with uniform density
RandExp	Generate random numbers with exponential density
RandNorm	Generate random numbers with normal density
Round	Round a real number to the nearest integral value
Sin	Sine of a number or array
Sinh	Hyperbolic sine of a number or array
Sqrt	Square root of a number or an array
Tan	Tangent of a number or array
Tanh	Hyperbolic tangent of a number or array
Trunc	Remove fractional part of number or array

Array and matrix arithmetic

These functions are used with arrays and result views to speed up data manipulation. You can also use the built-in mathematical functions directly on an array.

ArrAdd	Adds an array or constant to an array
ArrConst	Copies an array, or sets an array to a constant value
ArrDiff	Replaces an array with an array of simple differences
ArrDiv	Divides an array by another array or a constant
ArrDivR	Divides array into another array or constant
ArrDot	Forms the dot product (sum of products) of two arrays
ArrFFT	Fourier transforms and related operations
ArrFilt	Applies a FIR filter to an array
ArrHist	Calculate a histogram from a data array
ArrIntgl	Integrates array; inverse of ArrDiff()
ArrMul	Multiplies an array by another array or constant
ArrSort	Sort an array and optionally order others in the same way
ArrSpline	Interpolate one array to another using cubic splines
ArrStats	Calculate mean, variance, skewness and kurtosis of an array
ArrSub	Subtract constant from array, or difference of two arrays
ArrSubR	Subtract array from constant, or reversed difference of arrays
ArrSum	Sum, mean and standard deviation of an array
Len	Returns the length of a string or array

LinPred	Linear prediction and maximum entropy method power spectra
MATDet	Calculate the determinant of a matrix
MATInv	Invert a matrix
MATMul	Multiply matrices and vectors.
MATSolve	Solve a set of linear equations
MATTrans	Transpose a matrix (also see the trans() operator)
PCA	Principal component analysis (singular value decomposition)

String functions

These functions manipulate strings and convert between strings and other variable types.

Asc	ASCII code of first character of a string
Chr\$	Converts a code to a one character string
DelStr\$	Returns a string minus a sub-string
InStr	Searches for a string in another string
LCase\$	Returns lower case version of a string
Left\$	Returns the leftmost characters of a string
Len	Returns the length of a string or array
Mid\$	Returns a sub-string of a string
Print\$	Produce formatted string from variables
ReadStr	Extract variables from a string
ReadSetup	Set separators and delimiters for ReadStr() and Read()
Right\$	Returns the rightmost characters of a string
Str\$	Converts a number to a string
Trim	Remove leading and trailing white space or user-defined characters
TrimLeft	Remove leading white space or user-defined characters
TrimRight	Remove trailing white space or user-defined characters
UCase\$	Returns upper case version of a string
Val	Converts a string to number

Curve fitting

These functions do least squares fits of data to functions.

ChanFit	Fit function to time, result or XY channels and display fitted line
FitData	Fit a selected function to arrays of x,y data points
FitExp	Fits multiple exponentials
FitGauss	Fit multiple Gaussians (normal distributions)
FitLine	Fit straight line to time or result view data
FitLinear	Fit to linear combination of user-defined functions
FitNLUser	Fit to non-linear user function
FitPoly	Fit polynomial to the data
FitSigmoid	Fit sigmoid to the data
FitSin	Fit multiple sinusoids

[Click here to learn more about curve fitting.](#)

Digital filtering

These functions create and apply digital filters and manipulate the filter bank.

ArrFilt	Array arithmetic routine to apply FIR coefficients to an array
FiltApply	Apply a set of coefficients or a filter bank filter to a waveform
FiltAtten	Set the desired attenuation of an FIR filter in the filter bank
FiltCalc	Force coefficient calculation of an FIR filter in the filter bank
FiltComment\$	Get or set the comment for an FIR filter in the filter bank
FiltCreate	Create a new FIR filter definition in the filter bank
FiltInfo	Retrieve information about an FIR filter in the filter bank
FiltName\$	Get or set the name of an FIR filter in the filter bank

<code>FiltRange</code>	Get the useful sampling rate range for an FIR filter bank filter
<code>FIRMake</code>	Generate FIR filter coefficients in an array
<code>FIRQuick</code>	Generate FIR filter coefficients with desired attenuation
<code>FIRResponse</code>	Calculate frequency response of array of FIR coefficients
<code>IIRApply</code>	Apply an IIR filter bank filter to a waveform channel
<code>IIRBp</code>	Create and/or apply an IIR bandpass filter
<code>IIRBs</code>	Create and/or apply an IIR bandstop filter
<code>IIRComment\$</code>	Get or set the comment for an IIR filter in the filter bank
<code>IIRCreate</code>	Create a new IIR filter definition in the filter bank
<code>IIRHp</code>	Create and/or apply an IIR highpass filter
<code>IIRLp</code>	Create and/or apply an IIR lowpass filter
<code>IIRName\$</code>	Get or set the name of an IIR filter in the filter bank
<code>IIRNotch</code>	Create and/or apply an IIR notch filter
<code>IIRReson</code>	Create and/or apply an IIR resonator filter

Sampling configuration

These commands set the sampling configuration to use when you create a new data file.

<code>OutputReset</code>	Control reset values for DAC and digital outputs
<code>SampleAutoComment</code>	Control automatic prompt for file comment after sampling
<code>SampleAutoCommit</code>	Controls how frequently a data file is flushed to disk
<code>SampleAutoFile</code>	Set if file automatically written to disk at end of sampling
<code>SampleAutoName\$</code>	Set file name template for automatic file saving
<code>SampleBigFile</code>	Set or get the big file setting in the sampling configuration
<code>SampleCalibrate</code>	Set calibration of waveform or WaveMark channel
<code>SampleChanInfo</code>	Get information about channels in the sampling configuration
<code>SampleChannels</code>	Set or get the maximum number of channels to sample
<code>SampleComment\$</code>	Set or get channel comments or Script Bar comment and label
<code>SampleConfig\$</code>	Get the name of the last loaded configuration file
<code>SampleClear</code>	Sets the Sampling configuration to a known state
<code>SampleDebounce</code>	Get and set the debounce time for Event and digital marker channels
<code>SampleEvent</code>	Adds a channel to sample event data
<code>SampleDigMark</code>	Adds a channel to sample digital marker data
<code>SampleKeyMark</code>	Restore the keyboard marker channel to a sampling configuration
<code>SampleLimitSize</code>	Set or clear the size limit on a file
<code>SampleLimitTime</code>	Set or clear the time limit for sampling
<code>SampleMode</code>	Set the sampling mode (Continuous, Timed, Triggered)
<code>SampleOptimise</code>	Set methods for optimising waveform and WaveMark rates
<code>SampleRepeats</code>	Set the number of files to sample in sequence
<code>SampleSeqClock</code>	Get and optionally change the sequencer clock rate
<code>SampleSeqCtrl</code>	Set or get the allowed sequencer jump methods
<code>SampleSequencer</code>	Set the name of the sequencer file
<code>SampleSequencer\$</code>	Get the name of the sequencer file
<code>SampleStartTrigger</code>	Set triggered or non-triggered sample start
<code>SampleTextMark</code>	Adds a channel for text notes
<code>SampleTimePerAdc</code>	Set the base ADC conversion interval
<code>SampleTitle\$</code>	Set and get the title of a channel
<code>SampleTrigger</code>	Enable peri-trigger sampling of groups of channels
<code>SampleUsPerTime</code>	Set and get the basic timing interval
<code>SampleWaveform</code>	Adds a channel to sample waveform data
<code>SampleWaveMark</code>	Adds a channel to sample WaveMark data

Runtime sampling

These commands control data sampling. There is only one new data file at a time, and these commands refer to it, regardless of the current view.

<code>SampleAbort</code>	Stop sampling and throw data away
<code>SampleHandle</code>	Gets the view handle of sampling windows and controls
<code>SampleKey</code>	Adds to the keyboard marker channel, controls output sequencer

SampleReset	Clear the current data file and restart sampling
SampleSeqClock	Get and optionally change the sequencer clock rate
SampleSeqStep	Get the current sequencer step
SampleSeqTable	Get and set output sequencer table data
SampleSeqVar	Set an output sequencer variable
SampleSeqTick0()	Set/get the reference time for the TICKS instruction
SampleStart	Start sampling after creating a new time view
SampleStatus	Get the sampling state
SampleStop	Stop sampling and keep the data
SampleText	Adds a string to the text marker channel
SampleWrite	Control writing data to sampling file

Arbitrary waveform output

These commands control the output of waveforms during data sampling and offline. Waveforms are identified by a key code and up to 10 can be defined.

PlayOffline	Play offline via 1401 or sound card
PlayWaveAdd	Add waveform to the on-line play list
PlayWaveChans	Get or change the DAC channels of an area before sampling
PlayWaveCopy	Update output waveforms in 1401 after sampling starts
PlayWaveCycles	Get or change the repeats of a waveform
PlayWaveDelete	Delete one or more waveforms from the play wave list
PlayWaveEnable	Controls which waveforms in the list are available for playing
PlayWaveInfo\$	Get waveform information including list of all waveform keys
PlayWaveKey2\$	Set a second key to play an arbitrary waveform
PlayWaveLabel\$	Get or set the label associated with each waveform
PlayWaveLink\$	Get or set the links between waveforms
PlayWavePoints	Change the size of an area online
PlayWaveRate	Get or set the desired base wave replay rate
PlayWaveSpeed	Scale factor for the wave replay rate - can be used on-line
PlayWaveStatus\$	Get information about waveform output during sampling
PlayWaveStop	Stop the current output immediately or set one more cycle
PlayWaveTrigger	Get or set the trigger state for a waveform

Signal conditioner control

These functions control serial line controlled signal conditioners.

CondFilter	Get or set the conditioner low-pass or high-pass filter
CondFilterList	Get a list of possible low-pass or high-pass filter settings
CondGain	Get or set the conditioner gain
CondGainList	Get a list of the possible gains for the conditioner
CondGet	Get all the settings for one channel of the conditioner
CondOffset	Get or set the conditioner offset for a channel
CondOffsetLimit	Get or set the conditioner offset range for a channel
CondRevision\$	Get or set the conditioner offset for a channel
CondSet	Single call to set all channel parameters
CondSourceList	Get names of the signal sources available on the conditioner
CondType	Get the type of signal conditioner

Debugging operations

These functions can be used when debugging a script.

App(-4)	Get the number of system handles held by Spike2
Debug	Set a permanent break point and disable/enable debugging
DebugHeap()	Get information about Spike2 memory usage
DebugList	List internal Spike2 script objects
DebugOpts	Gets and optionally sets system level debugging options
Eval	Convert the argument to text and display

Environment

These functions don't fit well into any of the other categories!

App	Get the program serial number
Date\$	Get system date in a string in a variety of formats
Error\$	Convert a runtime error code to a message string
Profile	Access to the system registry and to Spike2 Preferences
ProgKill	Terminate a process started by ProgRun()
ProgRun	Run a program, optionally position the window
ProgStatus	Test if a program started by ProgRun() is still active
ScriptRun	Set a script to run when the current script ends
Seconds	Get or set current relative time in seconds
Sound	Play a tone or a .wav file
Speak	Convert text to speech on systems that support this
System	Get system revision as number
System\$	Get system name as a string, access environment variables
Time\$	Get system time in a string in a variety of formats
TimeDate	Get system date and time as numbers

Spike shape window

These functions can be used with a spike shape window. As the window is not a view in the sense of the time, result, XY and text views, to access the view you must use code of the form: `View(ss%).XLow()`; where `ss%` is the handle of the target spike shape window. To see examples, look in the scripts folder at `verifyss.s2s`, which tests script control of spike shape windows.

Cursor	Moving cursor 0 sets the time to read spikes from
FileClose	Close the spike shape window
HCursor	Set the horizontal cursor positions
Optimise	Optimise the y range and horizontal cursors
SSButton	Set and read button states in the window
SSChan	Set and get channel information, save channel configuration
SSClassify	Classify and create WaveMark data
SSOpen	Open spike shape windows and get handle of open window
SSParam	Equivalent to the spike shape parameters dialog
SSRun	Get and set the run state and speed
SSTempDelete	Delete templates based on index and code
SSTempGet	Read back template shape
SSTempInfo	Get and set template information (width, locking, counts)
SSTempSet	Create and modify templates
SSTempSizeGet	Get template size and offset in display and displayed points
SSTempSizeSet	Set template size and position and display size in points
ViewLink	Get the associated time view handle
Window	Set the position of the spike shape window
WindowGetPos	Get the window position
WindowVisible	Show and hide the spike shape window
XHigh	Get the start of the time view range to process
XLow	Get the end of the time view range to process
XRange	Set time range in associated time view to process
YHigh	Read back the y axis range
YLow	Read back the y axis range
YRange	Set displayed y range

Spike Monitor

These function control the Spike monitor window.

SMControl	Get and set the states of the control bar items
SMOpen	Open and or get the handle of a spike monitor window

ViewLink	Get the associated time view handle
Window	Set the position of the spike shape window
WindowGetPos	Get the window position
WindowVisible	Show and hide the spike shape window

Multimedia files

These function control avi files associated with data files.

MMAudio	Get information on the audio content
MMImage	Get a video image in a variety of formats
MMOpen	Open and or get the handle of a multimedia file
MMPosition	Set the play position and play state of a multimedia file
MMRate	Set the desired frames per second during recording
MMVideo	Get information about the video format
FileSaveAs	Save a video frame as an image

Serial line control

These functions let the script writer read to and write from serial line ports on their computer. This feature would normally be used to control equipment during data capture.

SerialOpen	Open a serial port and configure it (set Baud rate, parity etc.)
SerialWrite	Write characters to the serial port
SerialRead	Read characters from the serial port
SerialCount	Count the number of data items available to read
SerialClose	Release a previously opened serial port

Multiple monitor support

These functions give the script writer control over positioning the Spike2 application and Spike2 windows in a multiple monitor environment.

DlgCreate	Position a script-controllable dialog relative to monitors.
DlgMouse	Set mouse position relative to dialog (useful with multiple monitors)
System	Get monitor count, positions and identify the primary monitor.
Window	Position a script-controllable window relative to monitors.
WindowGetPos	Get position of a window, optionally including screen information.
WindowVisible	Maximise the application over the entire desktop.

1401 control

These functions communicate with a 1401 when Spike2 is not sampling. These functions will communicate with any 1401 that is supported by the 1401 device driver; it is not limited to the 1401 types that are supported for sampling in Spike2 version 8.

U1401Close	Release the 1401 for use by Spike2 sampling.
U1401Ld	Load 1401 commands into the 1401 from disk files.
U1401Open	Open a 1401 for use by the other U1401xxx commands.
U1401Read	Read a text response from a 1401, optionally convert to numbers.
U1401To1401	Transfer an integer array to the 1401 from the host.
U1401ToHost	Transfer an integer array to the host from the 1401
U1401Write	Write a text string to the 1401

Alphabetical script function index

This section is an alphabetical list of all implemented functions.

A

Abs()

This evaluates the absolute value of an expression as a real number. This can also form the absolute value of a real or integer array.

```
Func Abs(x|x[]|{[]...});
```

x A real number, or a real or integer array

Returns If x is an array, this returns 0 if all was well, or a negative error code if integer overflow was detected. Otherwise it returns x if x is positive, otherwise -x.

See also:

ATan(), Cos(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan(), Trunc()

App()

This returns view handles for system specific windows and program information.

```
Func App({type%});
```

type% Negative values return application specific information:

- 1 The program revision multiplied by 100.
- 2 The highest type% that returns a value.
- 3 The program serial number.
- 4 The number of system handles held by the application (for debugging) or 0 if this is not supported. Added at 6.08.
- 5 The number of GDI (graphic) objects used by the application (for debugging) or 0 if this is not supported. Added at 6.08.
- 6 Indicates if this is a 64-bit build of Spike2 (1) or a 32-bit build (0).

The remaining values return view handles. If type% is omitted, 0 is used.

0 Application	4 Edit toolbar	8 Sample control panel
1 System toolbar	5 Play wave bar	9 Sequence control panel
2 Status bar	6 Script bar	10 Sample Status bar
3 Running script	7 Sample bar	

Returns The requested information or a handle for the selected window. If the requested window does not exist, the return value is 0. It is possible for App(3) to return 0 if the running script has been unloaded from memory while it is executed. This can happen if it is started with ScriptRun().

Do not confuse the system toolbar with the script toolbar controlled with the Toolbar() command.

Example

The following can be used to hide all the floating windows at the start of a script and restore them at the end. The gFloat% array should be a global at the start of your script.

```
var gFloat%[20];                    'global for floating window states
proc HideAll()
var i%;
gFloat%[0] := App(-2);            'number of windows
```

```

for i% := 1 to gFloat%[0] do 'hide all windows and save state
  gFloat%[i%] := View(App(i%)).WindowVisible(0);
  next;
end
proc RestoreAll()
var i%;
for i% := 1 to gFloat%[0] do 'restore hidden windows
  View(App(i%)).WindowVisible(gFloat%[i%]);
  next;
end

```

See also:

View(), Dup(), LogHandle(), System(), Window()

ArrAdd()

This function adds a constant or an array to a real or integer array.

```
Func ArrAdd(dest[]{[]...}, source[]{[]...}|value);
```

dest The destination array (1 to 5 dimensions).

source An array of reals or integers with the same number of dimensions as **dest**. If the dimensions have different sizes, the smaller size is used for each dimension.

value A value to be added to all elements of the destination array.

Returns The function returns 0 if all was well, or a negative error code for integer overflow. Overflow is detected when adding a real array to an integer array and the result is set to the nearest valid integer.

In the following examples we assume that the current view is a result view:

```

var fred[100], jim%[200], two[3][30], add[3][30];
ArrAdd(fred[],1.0);          'Add 1.0 to all elements of fred
ArrAdd(fred[],jim%[]);      'Add elements 0-99 of jim% to fred
ArrAdd([],fred[10]);        'Add fred[10] to result channel 1
ArrAdd(two[][], add[]{});   'add corresponding elements

```

See also:

ArrConst(), ArrDiff(), ArrDiv(), ArrDivR(), ArrDot(), ArrFFT(), ArrFilt(), ArrIntgl(), ArrMul(), ArrSub(), ArrSubR(), ArrSum(), Len()

ArrConst()

This function sets an array or result view to a constant value, or copies the elements of an array or result view to another array or result view. You can copy number or strings. It is an error to attempt to copy numbers to a string array, or strings to a numeric array.

```
Func ArrConst(dest[]{[]...}, source[]{[]...}|value);
```

dest The destination array of 1 to 5 dimensions of any type (real, integer, string).

source An array with same number of dimensions as **dest**. If the arrays have different sizes, the smaller size is used for each dimension.

value A value to be copied to all elements of the destination array.

Returns The function returns 0, or a negative error code. If an integer overflows, the element is set to the nearest integer value to the result.

In the examples below the indices **j** and **i** mean repeat the operation for all values of the indices. **a1d** and **b1d** are vectors, **a2d** and **b2d** are matrices. The arrays and value must either both be numeric, or both be strings.

Function	Operation
ArrConst(a1d[], value);	a1d[i] := value
ArrConst(a1d[], b1d[]);	a1d[i] := b1d[i]
ArrConst(a2d[][], value);	a2d[j][i] := value
ArrConst(a2d[][], b2d[]{});	a2d[j][i] := b2d[j][i]

See also:

`ArrAdd()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`,
`ArrIntgl()`, `ArrMul()`, `ArrSub()`, `ArrSubR()`, `ArrSum()`, `Len()`

ArrDiff()

This function replaces an array or result view with an array of differences. You can use this as a crude form of differentiation, however `ArrFilt()` provides a better method.

```
Proc ArrDiff(dest[]);
```

`dest[]` A real or integer vector that is replaced its differences. The first element of `dest` is not changed.

The effect of the `ArrDiff()` function can be undone by `ArrIntgl()`. The following block of code performs the same function as `ArrDiff(work[])`:

```
var work[100],i%;  
for i%:=99 to 1 step -1 do work[i%] -= work[i%-1]; next;
```

See also:

`ArrAdd()`, `ArrConst()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`,
`ArrIntgl()`, `ArrMul()`, `ArrSub()`, `ArrSubR()`, `ArrSum()`, `Len()`

ArrDiv()

This function divides a real or integer array by an array or a constant. Use `ArrDivR()` to form the reciprocal of an array. Division by zero and integer overflow are detected.

```
Func ArrDiv(dest[]{[]...}, source[]{[]...}|value);
```

`dest` An array of real or integer values.

`source` An array of reals or integers with the same number of dimensions as `dest`, used as the denominator of the division. If the arrays have different sizes, the smaller size is used for each dimension.

`value` A value used as the denominator of the division.

Returns 0 or a negative error code if integer overflow or division by zero occurs.

If there was integer overflow when assigning the result to an integer array the result is set to the nearest allowed integer value. If division by zero occurs, the associated destination element is not changed.

The function performs the operations listed below. The indices `j` and `i` mean repeat the operation for all values of the indices. Both `a1d` and `b1d` are vectors, `a2d` and `b2d` are matrices. The arrays and `value` may be integer or real.

Function	Operation
<code>ArrDiv(a1d[], value);</code>	<code>a1d[i] := a1d[i] / value</code>
<code>ArrDiv(a1d[], b1d[]);</code>	<code>a1d[i] := a1d[i] / b1d[i]</code>
<code>ArrDiv(a2d[][][], value);</code>	<code>a2d[j][i] := a2d[j][i] / value</code>
<code>ArrDiv(a2d[][][], b2d[][][]);</code>	<code>a2d[j][i] := a2d[j][i] / b2d[j][i]</code>

See also:

`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`,
`ArrIntgl()`, `ArrMul()`, `ArrSub()`, `ArrSubR()`, `ArrSum()`, `Len()`

ArrDivR()

This function divides a real or integer array into an array or a constant.

```
Func ArrDivR(dest[]{[]...}, source[]{[]...}|value);
```

`dest` An array of reals or integers used as the denominator of the division and for storage of the result.

`source` A real or integer array with the same number of dimensions as `dest` used as the numerator of the

division. If the arrays have different sizes, the smaller size is used for each dimension.

value A value used as the numerator of the division.

Returns 0 or a negative error code if integer overflow or division by zero occurs.

If there was integer overflow when assigning the result to an integer array the result is set to the nearest allowed integer value. If division by zero occurs, the associated destination element is not changed.

The function performs the operations listed below. The indices *j* and *i* mean repeat the operation for all values of the indices. Both *a1d* and *b1d* are vectors, *a2d* and *b2d* are matrices. The arrays and *value* may be integer or real.

Function	Operation
<code>ArrDivR(a1d[], value);</code>	<code>a1d[i] := value / a1d[i]</code>
<code>ArrDivR(a1d[], b1d[]);</code>	<code>a1d[i] := b1d[i] / a1d[i]</code>
<code>ArrDivR(a2d[][][], value);</code>	<code>a2d[j][i] := value / a2d[j][i]</code>
<code>ArrDivR(a2d[][][], b2d[][][]);</code>	<code>a2d[j][i] := a2d[j][i] / b2d[j][i]</code>

See also:

`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDiv()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`, `ArrIntgl()`, `ArrMul()`, `ArrSub()`, `ArrSubR()`, `ArrSum()`, `Len()`

ArrDot()

This function multiplies a vector by another and returns the sum of the products (sometimes called the dot product). The vectors are not changed.

```
Func ArrDot(arr1[], arr2[]);
```

arr1 A real or integer vector.

arr2 A real or integer vector.

Returns The function returns the sum of the products of the corresponding elements of the two vectors.

See also:

`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrFFT()`, `ArrFilt()`, `ArrIntgl()`, `ArrMul()`, `ArrSub()`, `ArrSubR()`, `ArrSum()`, `Len()`

ArrFFT()

This command performs spectral analysis on a result view, or on an array of data. Variants of this command produce log amplitude, linear amplitude, power and relative phase as well as an option to window the original data. The command has the syntax:

```
Func ArrFFT(dest[], mode%);
```

dest A real vector to process. It should be a power of two points long, from 8 points upwards; the upper size is limited by available memory. If the number of points is not a power of two, the size is reduced to the next lower power of two points.

mode% The mode of the command, in the range 0 to 5. Modes are defined below.

Returns The function returns 0 or a negative error code.

This command uses real arithmetic to calculate the fast Fourier transform. This is more precise, but somewhat slower than the integer transform used by `SetPower()`.

Modes 1 and 3-5 take an array of data that is a set of equally spaced samples in some unit (usually time). If this unit is *xin*, the output is equally spaced in units of $1/xin$. In the normal case of input equally spaced in seconds, the output is equally spaced in 1/seconds, or Hz. If there are *n* input points, and the interval between the input points is *t*, the spacing between the output points is $1/(n*t)$. The transform assumes that the sampled waveform is composed of sine and cosine waves of frequencies: 0, $1/(n*t)$, $2/(n*t)$, $3/(n*t)$ up to $(n/2)/(n*t)$ or $1/(2*t)$.

Display of phase in result views

The phase information sits rather uncomfortably in a result view. When it is drawn, the x axis has the correct increment per bin, but starts at the wrong frequency. If you need to draw it, the simplest solution is to copy the phase information to bin 1 from bin $n/2+1$ and set bins 0 and $n/2$ to 0 (this destroys any amplitude information):

```
ArrConst([1:], [n/2+1:]); 'Copy phase to the start of the view
[0]:=0; [n/2]:=0;        'Set phase of the DC and Nyquist points
Draw(0, n/2+1);         'Display the phase
```

Mode 0: Window data

This mode is used to apply a raised cosine window to the data array. See `SetPower()` for an explanation of windows. The selected data is multiplied by a raised cosine hanning window of maximum amplitude 1.0, minimum amplitude 0.0. This window causes a power loss in the result of a factor of $3/8$.

You can supply your own window to taper the data, using the array arithmetic commands. The raised cosine is supplied as a general purpose window.

Mode 1: Forward FFT

This mode replaces the data with its forward Fast Fourier Transform. You would most likely use this to allow you to remove frequency components, then perform the inverse transform. The output of this mode is in two parts, representing the real and imaginary result of the transform (or the cosine and sine components). The first $n/2+1$ points of the result hold the amplitudes of the cosine components of the result. The remaining $n/2-1$ points hold the amplitudes of the sine components. In the case of an 8 point transform, the output has the format:

point	frequency	contents	point	frequency	contents
0	DC(0)	DC amplitude	4	$4/(n*t)$	Nyquist amplitude
1	$1/(n*t)$	cosine amplitude	5	$1/(n*t)$	sine amplitude
2	$2/(n*t)$	cosine amplitude	6	$2/(n*t)$	sine amplitude
3	$3/(n*t)$	cosine amplitude	7	$3/(n*t)$	sine amplitude

There is no sine amplitude at a frequency of $4/(n*t)$, the Nyquist frequency, as this sine wave would have amplitude 0 at all sampled points.

Mode 2: Inverse FFT

This mode takes data in the format produced by the forward transform and converts it back into a time series. In theory, the result of mode 1 followed by mode 2, or mode 2 followed by mode 1, would be the original data. However, each transform adds some noise due to rounding effects in the arithmetic, so the transforms do not invert exactly. One use of modes 1 and 2 is to filter data. For example, to remove high frequency noise use mode 1, set unwanted frequency bins to 0, and use mode 2 to reconstruct the data.

Mode 3: dB and phase

This mode produces an output with the first $n/2+1$ points holding the log amplitude of the power spectrum in dB, and the second $n/2-1$ points holding the phase (in radians) of the data. In the case of our 8 point transform the output format would be:

point	frequency	contents	point	frequency	contents
0	DC(0)	log amplitude in dB	4	$4/(n*t)$	log amplitude in dB
1	$1/(n*t)$	log amplitude	5	$1/(n*t)$	phase in radians
2	$2/(n*t)$	log amplitude	6	$2/(n*t)$	phase in radians
3	$3/(n*t)$	log amplitude	7	$3/(n*t)$	phase in radians

There is no phase information for DC or for the point at $4/(n*t)$. This is because the phase for both of these points is zero. If you want the phase in degrees, multiply by 57.3968 ($180^\circ/\pi$). The log amplitude is calculated by taking the result of a forward FFT (same as mode 1 above) and forming:

$$dB = 10.0 \text{ Log}(power)$$

The *power* is calculated as for Mode 5

Mode 4: Amplitude and phase

This mode produces the same output format as mode 3, but with amplitude in place of log amplitude. The amplitude is calculated by taking the result of a forward FFT (same as mode 1 above), and forming:

$$amplitude = \sqrt{\cos^2 + \sin^2} \quad \text{There is no sin component for the DC and Nyquist}$$

Mode 5: Power and phase

This mode produces the same output format as modes 2 and 3, but with the result as power. The sum of the power components is equal to the sum of the squares of the original data divided by the number of data points. The power is calculated by taking the result of a forward FFT (same as mode 1 above), and forming:

$$power = (\cos^2 + \sin^2) * 0.5 \quad \text{for all components except the DC and Nyquist}$$

$$power = DC^2 \text{ or Nyquist}^2 \quad \text{for the DC and Nyquist components}$$

You can compare the output of this mode with the result of `SetPower()`. If you have a waveform channel on channel 1 in view 1, and do the following:

```
var spView%, afView%;           'assume in a time view
spView% := SetPower(1,1024);    'select power spectrum
Process(0,1024*View(-1).Binsize(1)); 'process first 1024 points
WindowVisible(1);              'make window visible
afView% := SetAverage(1,1024,0); 'To copy 1024 points
Process(0,0);WindowVisible(1);  'show the data
View(afView%);                 'Move to the view holding data
ArrFFT([], 0);ArrFFT([], 5);    'Apply window, take power spectrum
Draw(0,500);Optimise(0);        'Show 500 bins of Power
View(spView%);                 'Look at SetPower() result
Draw(0,500);Optimise(0);        'Show same bins of power spectrum
```

The results are identical except that the `ArrFFT()` view is 3/8 of the amplitude of the view generated by `SetPower()`. The reason for the difference is that the `SetPower()` command compensates for the effect of the window it uses internally by multiplying the result by 8/3. To produce the same numeric result, multiply by 8.0/3.0 (do NOT use 8/3, which is dividing an integer by an integer and results in 2, not 2.6666666...).

See also:

`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`, `ArrIntgl()`, `ArrMul()`, `ArrSub()`, `ArrSubR()`, `ArrSum()`, `Len()`, `SetPower()`

ArrFilt()

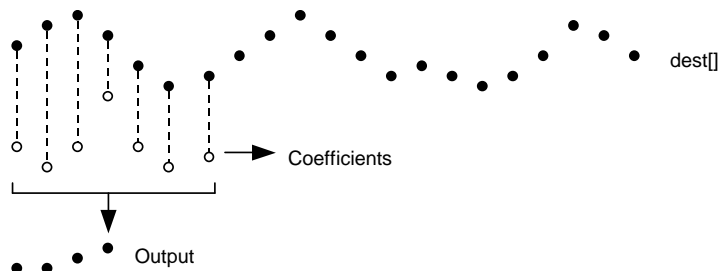
This function applies a FIR (Finite Impulse Response) filter to a real array. You can use `FiltCalc()` and `FIRMake()` to generate filter coefficients for a wide range of filters.

Func ArrFilt(dest[], coef[]);

dest[] A real vector holding the data to filter. It is replaced by the filtered data.

coef[] A real vector of filter coefficients. This is usually an odd number of data points long so that the result is not phase shifted. If you use an even number, the result is delayed by 1/2 a sample. Prior to Spike2 version 7.08, the result was advanced by 1/2 a sample. An even number of coefficients is used with a full differentiator.

Returns The function returns 0 if there was no error, or a negative error code.

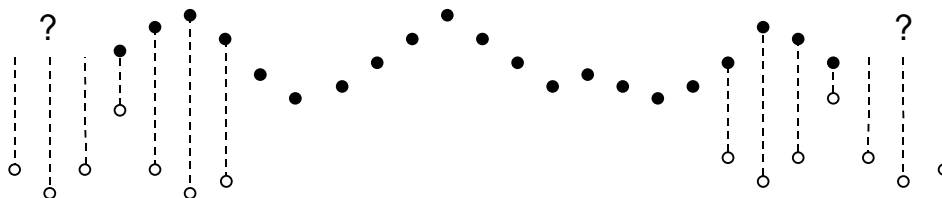


This diagram shows how the FIR filter works. Open circles represent filter coefficients, solid circles are the

input and output waveforms. Each output point is generated by multiplying the waveform by the coefficients and summing the result. The coefficients are then moved one step to the right and the process repeats.

From this description, you can see that the filter coefficients (from right to left) are the *impulse response* of the filter. The impulse response is the output of a filter when the input signal is all zero except for one sample of unit amplitude. In the example above with seven coefficients, there is no time shift in the output. If the filter has an even number of coefficients, there is an output time shift of half a sample.

The filter operation is applied to every vector element. There is a problem at the start and end of the vector where some coefficients have no corresponding data element.



The simple solution is to take these missing points as copies of the first and last points. This is usually better than taking these points as 0. You should remember that the first and last $(nc+1)/2$ points are unreliable, where nc is the number of coefficients.

A simple use of this command is to produce three point smoothing of data, replacing each point by the mean of itself and the two points on either side:

```
var data[1000],coef[3];      'Arrays of data and the coefficients
...                          'Fill data[] with values
coef[0]:=0.33333; coef[1]:=0.33333; coef[2]:=0.33333;
ArrFilt(data[],coef[]);     'smooth the data.
```

If you use this command to apply a causal filter, that is, one with all coefficients that use data points ahead of the current point set to zero, you must still provide these coefficients. If you omit the trailing zero coefficients, the output will be time shifted backwards by half the number of coefficients you do supply.

See also:

ArrAdd(), ArrConst(), ArrDiff(), ArrDiv(), ArrDivR(), ArrDot(), ArrFFT(), ArrIntgl(), ArrMul(), ArrSub(), ArrSubR(), ArrSum(), FiltCalc(), FIRMake(), Len()

ArrHist()

This function generates a histogram from a data array. This function is designed for repeated use with the same destination histogram and different source data; it is up to the caller to zero the histogram array before the first use. You can use `Min()`, `Max()`, `ArrSum()` and `ArrStats()` to find suitable values for the start and binSz parameters.

```
Func ArrHist(hist%[], data[], start, binSz{, &left%{, &right%}});
```

hist% An integer array to hold the generated histogram. The elements of this array are incremented by 1 for each item in the data array that falls in one of the histogram bins. The array is not set to zero before incrementing the bins. The bin to increment for data element `i%` is given by: $(data[i%]-start)/binSz$.

data An array of data values to be added into the histogram.

start The value at the start of the first bin,

binSz The width of each bin. This can be negative but must not be 0.

left% Optional. This integer variable is incremented by the number of data items with a negative bin number (falling to the left of the histogram).

right% Optional. This integer variable is incremented by the number of data items with a bin number greater than or equal to `Len(hist%[])` (falling to the right of the histogram).

Returns The number of data items that fell inside the histogram.

See also:

Min(), Max(), ArrSum(), ArrStats()

ArrIntgl()

This function is the inverse of `ArrDiff()`, replacing each point by the sum of the points from the start of the array up to the element. The first element is unchanged.

```
Proc ArrIntgl(dest[]);
```

`dest` A vector of real or integer data.

The function is equivalent to the following:

```
for i%:=1 to Len(dest[])-1 do dest[i%] += dest[i%-1]; next;
```

See also:

`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`, `ArrMul()`, `ArrSub()`, `ArrSubR()`, `ArrSum()`, `Len()`

ArrMul()

This command is used to form the product of a pair of arrays, or to scale an array by a constant. A less obvious use is to negate an array by multiplying by -1.

```
Func ArrMul(dest[]{[]...}, source[]{[]...}|value);
```

`dest` An array of real or integer numbers. If `dest` is integer, the multiplication is done as reals and truncated to integer.

`source` A real or integer array with the same number of dimensions as `dest`. If the arrays have different sizes, the smaller size is used for each dimension.

`value` A value to multiply the data in `dest`.

Returns The function returns 0 if all was well, or a negative error code.

The function performs the operations listed below. The indices j and i mean repeat the operation for all values of the indices. Both `a1d` and `b1d` are vectors, `a2d` and `b2d` are matrices. The arrays and `value` may be integer or real.

Function	Operation
<code>ArrMul(a1d[], value);</code>	<code>a1d[i] := a1d[i] * value</code>
<code>ArrMul(a1d[], b1d[]);</code>	<code>a1d[i] := a1d[i] * b1d[i]</code>
<code>ArrMul(a2d[][], value);</code>	<code>a2d[j][i] := a2d[j][i] * value</code>
<code>ArrMul(a2d[][], b2d[][]);</code>	<code>a2d[j][i] := a2d[j][i] * b2d[j][i]</code>

See also:

`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`, `ArrIntgl()`, `ArrSub()`, `ArrSubR()`, `ArrSum()`, `Len()`, `MATMul()`

ArrSort()

This function sorts an array of any data type and optionally orders additional arrays to match the sorted array.

```
Proc ArrSort(sort[]{, opt%{, arr1[]{, arr2[]{, ...}}});
```

`sort[]` An array of integer, real or string data to sort.

`opt%` This optional argument holds the sorting options. If omitted, the value 0 is used. It is the sum of the following flag values:

- 1 Sort in descending order. If omitted, the data is sorted in ascending order.
- 2 Case sensitive sort when `sort[]` is an array of strings. String sorts are usually case insensitive. If omitted, the sort is case insensitive.

`arrn[]` If present, these arrays are sorted in the same order as the `sort[]` array. The arrays can be of any

type. You can sort up to 18 additional arrays.

See also:

`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrMul()`,
`ArrSubR()`, `ArrSum()`, `Len()`

ArrSpline()

This function interpolates an array of real or integer data sampled at one rate into another array sampled at a different rate using cubic splines. It can also interpolate a matrix of (x,y) value pairs to an array sampled at a constant rate. This assumes that the source data has continuous first and second derivatives and that the second derivatives vary linearly from point to point. The second derivatives at the first and last point of the source data are set to zero. We interpolate from up to one input sampling interval before the first source point to up to one sampling interval after the last source point.

```
Func ArrSpline(dest[], source[]{, xInc{, xStart}});  

Func ArrSpline(dest[], srcXY[][]{, xInc{, xStart}});
```

dest A real or integer vector that holds the interpolated result. If the source array is a matrix, this is a real array.

source A vector of y values. The associated x values are assumed to be the same as the index of each data point (0, 1, 2...). It is integer array if **dest** is integer or real if **dest** is real.

srcXY A matrix of (x,y) source points. `srcXY[][0]` holds x values and `srcXY[][1]` holds y values. The x values must increase with the index, otherwise the return value is -1.

xInc If present, this is the x value change between **dest** values. If omitted, we assume that the points in **source** or **srcXY** and **dest** span the same time range. There are no restrictions on **xInc**, it can even be negative (in which case the output is backwards relative to the input). If the **source** variant is in use, **xInc** is the ratio of the **dest** sample interval to the **source** sample interval.

xStart If present, this is the x position of the first point of **dest**. If omitted, the first points of both arrays are at the same x position.

Returns 0 if all was OK or -1 if the **srcXY** variant is used and the x values do not increase monotonically.

You will get the best results if you can supply source data before and after the output time range. The effect of a source point on the interpolation of an interval n points away falls by a factor of approximately 4 each time n increases by 1. There is rarely the need to supply more than 15 data points before and after the interpolation range.

An example

Suppose we have a source vector of 100 points sampled at 100 Hz, with the first point sampled at 5.02 seconds, and we want to generate an array sampled at 1000 Hz that starts at 5.335 and lasts 0.5 seconds. In this case, the value of **xInc** is 0.001/0.01, which is 0.1. The value of start is 31.5, which is the time difference (5.335 - 5.02) divided by 0.01, the sample interval of the source channel.

See also:

`ChanProcessAdd()`, `DrawMode()`

ArrStats()

This calculates the mean, variance, skewness and (excess) kurtosis of an array of real data. It can also be used with a matrix to calculate vectors of the mean, variance, skewness and kurtosis of each column of the input matrix. Use the `ArrSum()` command if you only require the mean and standard deviation.

```
Func ArrStats(data[], &mean, &v, &skew, &kur);
```

data A vector of source data.

mean A real variable returned holding the mean of the data.

v A real variable returned holding the variance of the data.

`skew` A real variable returned holding the skewness of the data.

`kur` A real variable returned holding the kurtosis of the data.

Returns 0 if all was well or 1 if the variance of the data was 0.0 (in which case the skewness and kurtosis are also set to 0.0).

The second form of the command is provided as a convenience when the data is in a matrix with the data in the columns. When used in the second form, the vector must be at least the size of the first dimension of the matrix.

```
Func ArrStats(data[0][], mean[], v[], skew[], kur[]);
```

`data` A matrix with the data in the columns.

`mean` A real vector returned holding the means of each column of data.

`v` A vector returned holding the variances of each column of data.

`skew` A vector returned holding the skewness of each column of data.

`kur` A vector returned holding the kurtosis of each column of data.

Returns The number of columns for which the variance was 0 (in which case the skewness and kurtosis are also 0).

The first form of the command is equivalent to the following script (but runs many times faster):

```
Func ArrStats(arr[], &mean, &dVar, &skew, &kur)
var i%, n%;
mean := 0; dVar := 0; skew := 0; kur := 0;
ArrSum(arr, mean); 'Calculate the mean
n% := Len(arr);
var d, p;
for i% := 0 to n%-1 do
  d := arr[i%]-mean; p := d*d;
  dVar += p;          p *= d;
  skew += p;
  kur += p*d;
next;
if (dVar > 0.0) then 'If not a degenerate case...
  dVar /= (n%-1); 'calculate the variance
  skew /= n%*dVar*sqrt(dVar);
  kur /= n%*dVar*dVar;
  kur -= 3.0;
  return 0;
else
  return 1;
endif;
end;
```

The second form of the command is equivalent to:

```
var i%, return% := 0;
for i% := 0 to Len(data[0][])-1 do
  return% += ArrStats(data[0][i%], mean[i%], v[i%], skew[i%], kur[i%]);
next;
```

See also:

`ArrSum()`

ArrSub()

This function forms the difference of two arrays or subtracts a constant from an array. If the destination is an integer array, overflow is detected when subtracting real values.

```
Func ArrSub(dest[]{[]...}, source[]{[]...}|value);
```

`dest` A real or integer array that holds the result.

source A real or integer array with the same number of dimensions as **dest**. If the arrays have different sizes, the smaller size is used for each dimension.

value A real or integer value.

Returns 0 if all is well or a negative error code if integer overflow is detected.

The function performs the operations listed below. The indices j and i mean repeat the operation for all values of the indices. Both **a1d** and **b1d** are vectors; **a2d** and **b2d** are matrices. The arrays and **value** may be integer or real.

Function	Operation
<code>ArrSub(a1d[], value);</code>	<code>a1d[i] := a1d[i] - value</code>
<code>ArrSub(a1d[], b1d[]);</code>	<code>a1d[i] := a1d[i] - b1d[i]</code>
<code>ArrSub(a2d[][] , value);</code>	<code>a2d[j][i] := a2d[j][i] - value</code>
<code>ArrSub(a2d[][] , b2d[][]);</code>	<code>a2d[j][i] := a2d[j][i] - b2d[j][i]</code>

See also:

`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`, `ArrIntgl()`, `ArrMul()`, `ArrSubR()`, `ArrSum()`, `Len()`

ArrSubR()

This function forms the difference of two arrays or subtracts an array from a constant. If the destination is an integer array, overflow is detected when subtracting real values.

```
Func ArrSubR(dest[]{[]...}, source[]{[]...}|value);
```

dest A real or integer array.

source A real or integer array with the same number of dimensions as **dest**. If the arrays have different sizes, the smaller size is used for each dimension.

value A real or integer value.

Returns 0 if all is well or a negative error code if integer overflow is detected.

The function performs the operations listed below. The indices j and i mean repeat the operation for all values of the indices. Both **a1d** and **b1d** are vectors; **a2d** and **b2d** are matrices. The arrays and **value** may be integer or real.

Function	Operation
<code>ArrSubR(a1d[], value);</code>	<code>a1d[i] := value - a1d[i]</code>
<code>ArrSubR(a1d[], b1d[]);</code>	<code>a1d[i] := b1d[i] - a1d[i]</code>
<code>ArrSubR(a2d[][] , value);</code>	<code>a2d[j][i] := value - a2d[j][i]</code>
<code>ArrSubR(a2d[][] , b2d[][]);</code>	<code>a2d[j][i] := b2d[j][i] - a2d[j][i]</code>

See also:

`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`, `ArrIntgl()`, `ArrMul()`, `ArrSub()`, `ArrSum()`, `Len()`

ArrSum()

This function forms the sum of the values in a vector or matrix, and optionally forms the mean and standard deviation of the vector or of each matrix column. The standard deviation of m data points with a sum of squared difference from the mean of `errSq` is `sqrt(errSq/(m-1))`. There are two command variants:

```
Func ArrSum(arr[]|arr%[]{, &mean{, &stDev}});  
Func ArrSum(arr[][]|arr%[][]{, mean[], stDev[]});
```

arr A real or integer vector or matrix to process.

mean If present it is returned holding the mean of the values in the array. The mean is the sum of the values divided by the number of array elements. If **arr** is an m by n matrix, **mean** must be a vector of at least n elements and is returned holding the mean of each column of **arr**.

stDev If present, this returns the standard deviation of the array elements around the mean. If the array has only one element the result is 0. If `arr` is a `m` by `n` matrix, `stDev` must be a vector with at least `n` elements and is returned holding the standard deviation of each column of `arr`.

Returns The function returns the sum of all the array elements as a real number.

See also:

`ArrAdd()`, `ArrConst()`, `ArrDiff()`, `ArrDiv()`, `ArrDivR()`, `ArrDot()`, `ArrFFT()`, `ArrFilt()`, `ArrIntgl()`, `ArrMul()`, `ArrStats()`, `ArrSub()`, `ArrSubR()`, `Len()`

Asc()

This function returns the ASCII code of the first character in the string as an integer.

```
Func Asc(text$);
```

`text$` The string to process.

See also:

`Chr$()`, `DelStr$()`, `LCase$()`, `Left$()`, `Len()`, `Mid$()`, `Print$()`, `Right$()`, `Str$()`, `UCase$()`, `Val()`

ATan()

This function returns the arc tangent of an expression, or the arc tangent of an array:

```
Func ATan(s|s[]{|...} {,c|c[]{|...}});
```

`s` A value or an array of real values with any number of dimensions.

`c` Optional. If present, both `s` and `c` must be values, or both must be real arrays with the same number of dimensions. If `c` is present, `s/c` is a tangent and the result is in the range $-\pi$ to π using the signs of `s` and `c` to determine the quadrant. If `c` is omitted, the result is in the range $-\pi/2$ to $\pi/2$, equivalent to `c` being unity.

Returns If `s` is an array, each element of `s` (that is matched by an element of `c` if `c` is present) is replaced by the arc tangent. The function returns 0 if all was well or a negative error code. When `s` is not an array, the return value is the arc tangent of `s` or of `s/c` if `c` is present.

You can use `ATan()` to calculate other mathematical values. Arc sine of a single value: `s` can be calculated as: `ATan(s/Sqrt(1-s*s))`. Arc cosine can be calculated as: `ATan(Sqrt(1-s*s)/s)`.

Examples

```
var s[100], a[20][200], b, c[100], d[20][20];
ATan(s, c);           'both arguments are vectors
ATan(a);             'single argument is multi-dimensioned array
ATan(b);             'single argument is a variable
ATan(Sin(1), Cos(1)); 'result should be 1
ATan(a, d);         'Changes a[0:20][0:20] only (to match d)

Func ASin(s)         'Arc Sine function
return ATan(s, Sqrt(1-s*s));
end

Func ACos(s)         'Arc Cosine function
return ATan(Sqrt(1-s*s), s);
end
```

See also:

`Abs()`, `Cos()`, `Exp()`, `Frac()`, `Ln()`, `Log()`, `Max()`, `Min()`, `Pow()`, `Rand()`, `Round()`, `Sin()`, `Sqrt()`, `Tan()`, `Trunc()`

B

BetaI()

This function computes both the Beta function and the Incomplete Beta function. The Beta function is defined as:

$$\text{Beta}(a,b) = \int_0^1 t^{a-1}(1-t)^{b-1} dt$$

The incomplete beta function is defined as:

$$\text{BetaI}_x(a,b) = \frac{\int_0^x t^{a-1}(1-t)^{b-1} dt}{\text{Beta}(a,b)}$$

These functions are not usually directly of interest, but they are used in generating distribution functions, for example Student's distribution, the F-Distribution and the Binomial distribution.

```
Func BetaI(a, b{, x});
```

a, b These values must be greater than zero.

x Optional. If present, the result is the incomplete beta function. If omitted, the result is the beta function. x must be in the range 0 to 1.

Returns If x is present, the result is the incomplete beta function in the range 0 to 1. Otherwise, the result is the beta function.

See also:

GammaP(), GammaQ(), Binomial Distribution, Student's T Distribution, F-Distribution

Binomial Distribution

The Binomial distribution is used when you have repeated trials and for each trial an event has a known probability p of occurrence. The examples below show you how to calculate the probabilities that you will get $k\%$ or fewer events in $n\%$ trials and $k\%$ or more in $n\%$ trials.

```
'p        The probability per trial of the event
'k%       A number of events in the range 0 to n%
'n%       The number of trials (greater than 0)
'Return the probability that an event occurs k% or fewer times in n% trials.
Func BinomialLE(p, k%, n%)
if k% >= n% then return 1.0 endif;
if k% < 0 then return 0.0 endif;
return BetaI(n%-k%, 1+k%, 1-p);
end;
```

```
'p        The probability per trial of the event
'k%       A number of events in the range 0 to n%
'n%       The number of trials (greater than 0)
'Return the probability that an event occurs k% or more times in n% trials.
Func BinomialGE(p, k%, n%)
if (k% <= 0) then return 1.0 endif;
if (k% > n%) then return 0.0 endif;
return BetaI(k%, n%-k%+1, p);
end;
```

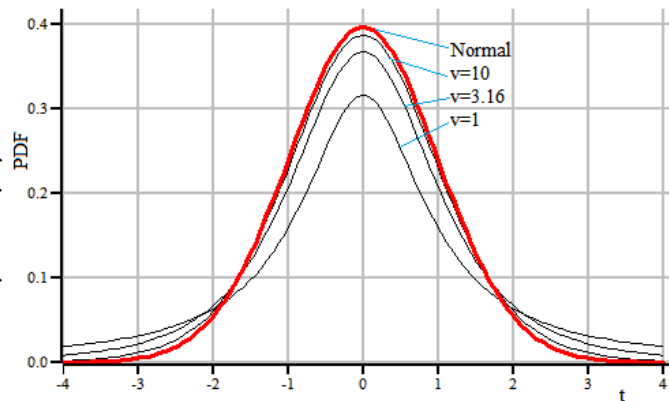
The BinomialC() function will evaluate binomial coefficients for you, avoiding arithmetic overflow as much as is possible.

See also:

BetaI(), BinomialC(), Student's T Distribution, F-Distribution

Student's T Distribution

The Student's T distribution is a symmetric probability density function (PDF) used when asking questions about the mean of a sample taken from a normally distributed population when the number of samples is small. It is characterised by the number of degrees of freedom (which is usually the number of samples used to estimate the mean minus 1), and becomes indistinguishable from the Normal distribution when the degrees of freedom reaches a hundred or so. If you want to graph the function, you can do so with this script expression:



```
const pi := 3.141592653589793;
func StudentsTDist(t, v)
return pow(1+t*t/v, -(v+1)/2)*Exp(LnGamma((v+1)/2) - LnGamma(v/2))/Sqrt(v*pi);
end;
```

The t value is a statistic that relates to how different two means are and can be positive or negative. Exactly how it is calculated depends on the situation; we give some examples later in this section, but the simplest case is where you have a normally distributed population and you take single sample of $n\%$ items with a sample mean S_m and sample standard deviation S_{sd} . If the true mean of the population is mean, the t statistic is:

$$t := (S_m - \text{mean}) * \text{Sqrt}(n\%) / S_{sd};$$

When asking questions about how likely one would be to observe a mean value that is different from or greater than or less than some value, the more useful functions are the cumulative probability distribution functions (in the range 0 to 1 as t varies from minus infinity to plus infinity), and the complement of this (1 minus it). These are usefully calculated from the incomplete beta function, as follows:

```
't The t statistic
'v The number of degrees of freedom (>0)
Func StudentCDF(t, v)
var tt := t*t;
var z := BetaI(0.5*v, 0.5, v/(v+tt))*0.5;
return t < 0 ? z : 1-z;
end;
```

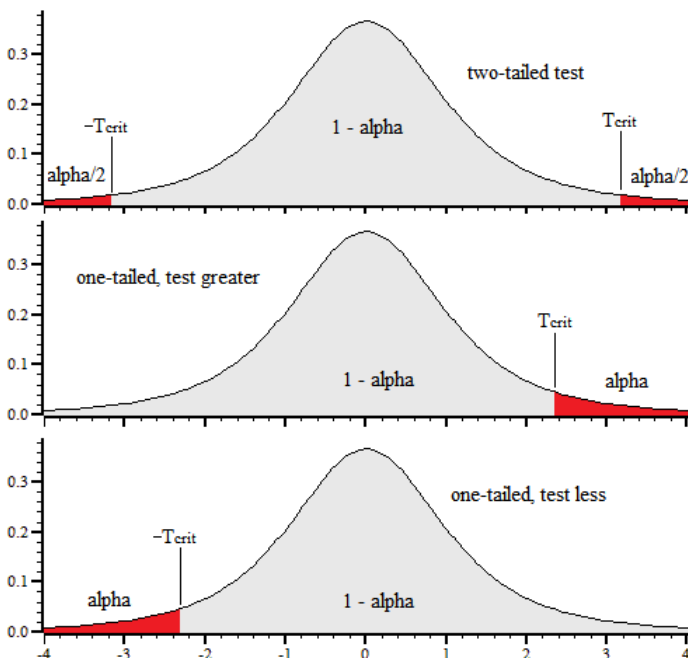
```
'This is 1.0-StudentCDF(t, v);
Func OneMinusStudentCDF(t, v)
var tt := t*t;
var z := BetaI(0.5*v, 0.5, v/(v+tt))*0.5;
return t < 0 ? 1-z : z;
end;
```

The `StudentCDF()` function returns the probability that one can reject the Null hypothesis that the means are not different (or not greater than or not less than some value). The value we usually want is `1-StudentCDF()`, which we calculate separately to preserve the accuracy for small values. In statistical tests, the probability level we are interested in, for example 0.05 (5%) or 0.01 (1%) is often referred to as *alpha*. `StudentCDF()` can be compared with `1-alpha`, which is called the confidence level (95% or 99%), `OneMinusStudentCDF()` returns the probability that the hypothesis being tested would occur by chance and so can be rejected. That is, you would reject the hypothesis if `OneMinusStudentCDF() > alpha`.

One-tail and two-tail tests

If you are asking the question, "are these two mean values different", you want to know how likely is a sampled mean value to occur by chance. If you want to be 99% certain that your sample differs, your t value needs to differ from 0 by a sufficient value that the cumulative probability density has reached a value such that 99% of the PDF lies at lesser values. As the PDF is symmetric about 0, this happens when 1% of the PDF is excluded, which happens when 0.5% is above some critical t value T_{crit} and 0.5% is below $-T_{crit}$. In this case your alpha is 0.01 (or 1%) and the value you would test for is $\alpha/2$. This is a two-tail test as you care what happens at both ends of the probability distribution.

If you are asking the question, "is this mean greater than that mean", you are only looking at one end of the distribution and the value you would test against is alpha. If you want to be 99% certain, T_{crit} is now the level at which 99% of the distribution lies below, your alpha value is 0.01 and this is the value to test for. Due to the symmetry of the distribution, to test that "this mean is less than that mean" you need a negative t value; the probability to test for is the same as for a positive t value. A one-tail test should be used with caution.



Inverse of the distribution

The functions above calculate alpha given a t statistic and the number of degrees of freedom. If you want to specify alpha and calculate the t statistic value that corresponds to this, you need the inverse function. The following script code will do this for you:

```
'Function used by ZeroRoot() to locate the t value with the required probability
var gProb, gV;      'probability and degrees of freedom
func TZeroRoot(t)
return gProb - OneMinusStudentCDF(t, gV);
end;

'TValueFor: This calculates the t-value that you need to reject the NULL hypothesis
'given an alpha, the number of degrees of freedom and if you want a one sided test.
'alpha The desired probability level (typically 0.05)
'v      Degrees of freedom (> 0.0)
'os%   0=two sided, non-zero for 1 sided
func TValueFor(alpha, v, os%)
var t;
gProb := os% ? alpha : alpha*0.5;
gV := v;
ZeroFind(t, TZeroRoot, -20, 20);
return t;
end;
```

Examples

The following example code follows examples given for the Boost C++ libraries, which in turn follow NIST/SEMATECH e-Handbook of Statistical Methods, <http://www.itl.nist.gov/div898/handbook/> (NIST is an agency of the US Commerce Department).

Confidence range of the mean

You have $S_n\%$ sample values taken from a normally distributed population with mean S_m and standard deviation S_{sd} and you want to know in what range of values can you be confident that you have bracketed the true population mean. This is plainly a two-tail test as we care about values in both directions. If you want to be certain at the alpha level, the T_{crit} value is given by: $TValueFor(\alpha/2, S_n\%-1, 0)$ and the range is from $S_m - T_{crit} * S_{sd} / \sqrt{S_n\%}$ to $S_m + T_{crit} * S_{sd} / \sqrt{S_n\%}$. See here for details.

Test a sample mean for difference from a known mean

This test comes up if you already know what the mean should be (for example in quality control) and you want to know if there is a change. It can also come up in a paired test where you measure a value before and after a treatment on a set of items and your data set is the difference between before and after for each item, so you are asking is there a difference from zero. The following routine does this calculation:

```
'testing sample mean for difference from a "true" mean.
'n%      Number of items in the sample
'Sm      Sample mean
'Ssd     Sample standard deviation
'mean    The "true" mean
'os%     -1 means one-sided test for below mean, +1 means one-sided test for
'         above the mean, 0 means two-sided test for different
'Return  the probability that the Null hypothesis (that the sample mean is
'         not different, greater or less than the true mean).
Func TSingleSample(n%, Sm, Ssd, mean, os%)
var t := (Sm - mean)* Sqrt(n%)/Ssd;
PrintLog("\nStudent's t test for a single sample\n");
PrintLog("Number of observations      = %D\n", n%);
PrintLog("Sample mean                  = %g\n", Sm);
PrintLog("Sample Standard Deviation      = %g\n", Ssd);
PrintLog("Expected True mean              = %g\n", mean);
PrintLog("T Statistic                      = %g\n", t);
PrintLog("Degrees of freedom               = %d\n", n%-1);
if (os% = 0) then t := abs(t) endif;
if (os% < 0) then t := -t endif;
var prob := OneMinusStudentCDF(t, n%-1); 'Single tailed value
if (os% = 0) then prob *= 2.0 endif;
PrintLog("Probability due to chance      = %g\n", prob);
return prob;
end;
```

In this test we have a single sample of $n\%$ items with mean S_m (Sample mean) and standard deviation S_{sd} (Sample standard deviation). The question is: is the mean different (two-tail) or greater or less (one-tail) than the known mean. The return value is the probability that the Null hypothesis can be rejected and is the value to compare against the alpha value for your test. You may choose to omit the `PrintLog()` lines. To use this function, for example using data from the NIST site: 3 observations with mean of 37.8, standard deviation of 0.96437 and expected true mean of 38.9:

```
TSingleSample(3, 37.8, 0.96437, 38.9, -1); 'Test for mean < 38.9
TSingleSample(3, 37.8, 0.96437, 38.9, 0); 'Test for mean <> 38.9
TSingleSample(3, 37.8, 0.96437, 38.9, 1); 'Test for mean > 38.9
```

The output is (omitting repeated lines for second and third cases):

```
Student's t test for a single sample
Number of observations      = 3
Sample mean                = 37.8
Sample Standard Deviation  = 0.96437
Expected True mean        = 38.9
T Statistic                = -1.97565
Degrees of freedom        = 2
Probability due to chance   = 0.093429 (Hypothesis: mean < 38.9)
...
Probability due to chance   = 0.186858 (Hypothesis: mean <> 38.9)
...
Probability due to chance   = 0.906571 (Hypothesis: mean > 38.9)
```

With an alpha of 0.05 (5%), all the hypotheses are rejected. However, if the alpha level was set at 0.1 (10%), then the hypothesis that the mean is less than 38.9 is not rejected.

Test two samples with equal variance for difference of means

In this case we have a first sample with $n1\%$ items with mean $S1m$ and standard deviation $S2sd$ and a second sample with $n2\%$ items and mean $S2m$ and standard deviation $S2sd$. Again, we want to ask if the means are different, or if the first mean is less than or greater than the second. The code to do this is (you can delete the `PrintLog()` lines if they are not wanted):

```
'ni%     The number in each set of samples
'Sim     The mean of sample i
'Sisd    The standard deviation of sample i
```

```

'os% 0 for two sided. 1 for sm1 > sm2, -1 for sm1 < sm2
'Return the probability that the Null hypothesis (that the sample means
' are not different, S1m1 greater or less than S2m2).
Func TTwoSamplesSameVar(n1%, S1m, S1sd, n2%, S2m, S2sd, os%)
'Calculate the combined standard deviation of both samples
var S12sd := Sqrt(((n1%-1)*S1sd*S1sd + (n2%-1)*S2sd*S2sd)/(n1%+n2%-2));
var t := (S1m - S2m)/(S12sd * Sqrt(1.0/n1% + 1.0/n2%));
PrintLog("\nStudent's t test for two samples, same variance\n");
PrintLog("Number of observations = %8d %8d\n", n1%, n2%);
PrintLog("Sample means = %8g %8g\n", S1m, S2m);
PrintLog("Sample Standard Deviation = %8g %8g\n", S1sd, S2sd);
PrintLog("T Statistic = %g\n", t);
PrintLog("Degrees of freedom = %d\n", n1%+n2%-2);
if (os% = 0) then t := abs(t) endif;
if (os% < 0) then t := -t endif;
var prob := OneMinusStudentCDF(t, n1%+n2%-2); 'One-tail value
if (os% = 0) then prob *= 2.0 endif;
PrintLog("Probability due to chance = %8e\n", prob);
return prob;
end;

```

As the two samples have the same variance we can form a pooled standard deviation by combining the two standard deviations, which is what the calculation for `S12sd` does. The number of degrees of freedom is just the sum of the degrees of freedom of the two data sets. The calculation of the `t` statistic is then very similar to the previous single sample test, just weighting by the two sample sizes. As an example, we consider the following two data sets:

Data set	Observations	Mean	Sd
1	249	20.1446	6.4147
2	79	30.481	6.10771

The code to test each possible hypothesis is:

```

TTwoSamplesSameVar(249, 20.14458, 6.41470, 79, 30.48101, 6.10771, -1); 'mean 1 < mean 2
TTwoSamplesSameVar(249, 20.14458, 6.41470, 79, 30.48101, 6.10771, 0); 'mean 1 <> mean 2
TTwoSamplesSameVar(249, 20.14458, 6.41470, 79, 30.48101, 6.10771, 1); 'mean 1 > mean 2

```

The output (omitting duplicated lines is):

```

Student's t test for two samples, same variance
Number of observations = 249 79
Sample means = 20.1446 30.481
Sample Standard Deviation = 6.4147 6.10771
T Statistic = -12.6206
Degrees of freedom = 326
Probability due to chance = 2.636609e-030 (Hypothesis: mean1 < mean 2)
...
Probability due to chance = 5.273218e-030 (Hypothesis: mean1 <> mean 2)
...
Probability due to chance = 1.000000e+000 (Hypothesis: mean 1 > mean 2)

```

This says that the hypotheses that `mean1` is less than `mean2` or that `mean1` is different from `mean2` cannot be rejected. The hypothesis that `mean1` is greater than `mean2` is rejected.

Test two samples with non-equal variance

Finally we have the code for two samples with non-equal variance. This time we cannot pool the standard deviations, and we also have a problem with the degrees of freedom. To calculate this, the Welch-Scatterthwaite approximation is used. Note that the approximation improves as the number of degrees of freedom increases. As ever, you can delete all the `PrintLog()` lines if they are not required:

```

Func TTwoSamplesDiffVar(n1%, S1m, S1sd, n2%, S2m, S2sd, os%)
var s1 := S1sd*S1sd/n1%;
var s2 := S2sd*S2sd/n2%;
'Calculate the t value
var t := (S1m - S2m)/Sqrt(s1 + s2);
'Calculate the combined degrees of freedom using the Welch-Scatterthwaite approximation.
var v := (s1+s2)*(S1+S2)/(s1*s1/(n1%-1) + s2*s2/(n2%-1));
PrintLog("\nStudent's t test for two samples, different variance\n");
PrintLog("Number of observations = %8d %8d\n", n1%, n2%);
PrintLog("Sample means = %8g %8g\n", S1m, S2m);
PrintLog("Sample Standard Deviation = %8g %8g\n", S1sd, S2sd);
PrintLog("T Statistic = %g\n", t);

```

```
PrintLog("Degrees of freedom      = %g\n", v);
if (os% = 0) then t := abs(t) endif;
if (os% < 0) then t := -t endif;
var prob := OneMinusStudentCDF(t, v);   'One-tail value
if (os% = 0) then prob *= 2.0 endif;    'Convert to two-tail value
PrintLog("Probability due to chance = %8e\n", prob);
return prob;
end;
```

If we use the same input data as for the previous example, we have:

```
TTwoSamplesDiffVar(249, 20.14458, 6.41470, 79, 30.48101, 6.10771, 0); 'mean 1 < mean 2
TTwoSamplesDiffVar(249, 20.14458, 6.41470, 79, 30.48101, 6.10771, 0); 'mean 1 <> mean 2
TTwoSamplesDiffVar(249, 20.14458, 6.41470, 79, 30.48101, 6.10771, 0); 'mean 1 > mean 2
```

The output (omitting repeated lines) is:

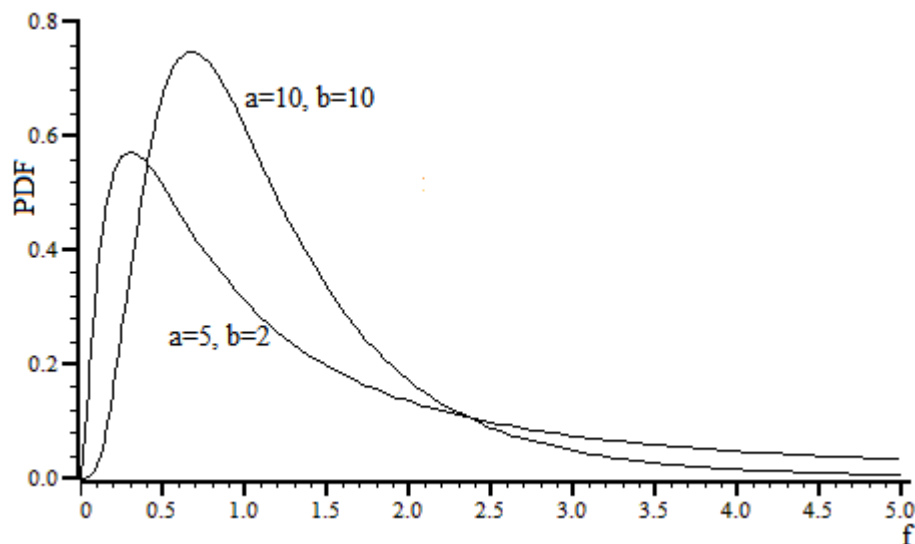
```
Student's t test for two samples, different variance
Number of observations      =      249      79
Sample means                =  20.1446   30.481
Sample Standard Deviation  =   6.4147   6.10771
T Statistic                 = -12.9463
Degrees of freedom         =  136.875
Probability due to chance   =  7.854523e-026 (Hypothesis: mean1 < mean 2)
...
Probability due to chance   =  1.570905e-025 (Hypothesis: mean1 <> mean 2)
...
Probability due to chance   =  1.000000e+000 (Hypothesis: mean1 > mean 2)
```

In this case, although the probability that this occurred by chance is increased by several orders of magnitude compared to the equal variance situation, the conclusions are the same.

See also:

BetaI(), Binomial Distribution, F-Distribution

F-Distribution



F Distribution Probability Density Function

The F Distribution probability density functions shows the likelihood of particular values of the F statistic occurring. The F statistic calculation depends on the situation; in a simple case it is the ratio of two variances. The F distribution can be calculated and displayed by the script code:

```
'The derivative of the BetaI function, used to calculate the Prob density of
'the F-Distribution. Maths as suggested by the Boost library.
Func BetaIDerivative(a, b, x)
if (a<=0.0) or (b<=0.0) or (x<0) or (x > 1.0) then return -1 endif;
if (x=0.0) or (x=1.0) then return 0 endif;
return exp((b-1)*ln(1-x)+(a-1)*ln(x))/BetaI(a,b);
end;
```

```
'The Probability Density function for the F-Distribution. The use of two
'methods keeps the third argument to the derivative function away from 1.
Func FDistribution(a, b, x)
var z := b + a * x;
var y := a * b / (z * z);
if (a*x > b) then
    return y * BetaIDerivative(b/2, a/2, b/z);
else
    return y * BetaIDerivative(a/2, b/2, a*x/z);
endif;
end;

'Draw the F distribution from x=0 to range with a and b degrees of freedom
Func ShowFDist(a, b, range)
const n% := 100;          'points after 0
const np% := n%+1;       'total points to plot
var rv% := SetResult(np%, range/n%, 0, "F distribution", "f", "PDF");
DrawMode(-1,13);        'cubic spline mode
var i%, x;
for i% := 0 to n% do
    x := i%*(range/n%);
    [i%] := FDistribution(a, b, x);
next;
WindowVisible(1);
return rv%;
end;

ShowFDist(5, 2, 5); 'Draw an example
```

Unlike the Student's T distribution, the F distribution is asymmetric. For us, a more interesting and useful function is the Cumulative Distribution Function (CDF) of the F distribution. For statistical use, the complement of the CDF (1-CDF) is usually the most useful. The CDF and its complement can be calculated with:

```
'The CDF for the F-Distribution.
func FDistCDF(a, b, x)
var ax := a*x;
var z := b+ax;
return BetaI(a/2, b/2, ax/z);
end;

'We usually want the complement of the CDF
func OneMinusFDistCDF(a, b, x)
var ax := a*x;
var z := b+ax;
return BetaI(b/2, a/2, b/z);
end;
```

The value of the `FDistCDF()` function increases from 0 to 1 as x (the F statistic) varies between 0 and infinity. The value of `OneMinusFDistCDF()` decreases from 1 to 0 over the same range.

Inverse of the F distribution CDF complement

When asking questions such as, "What F value does 5% of the F-Distribution lie above?", we find the inverse of the complement of the F distribution CDF at the 5% level. This can be calculated by the `InvFCDFComp()` function:

```
var gA, gB, gAlpha; 'variables used by InvFCDFComp()

'Helper function for InvFCDFComp(), used by ZeroFind()
Func ZeroFRoot(x)
return OneMinusFDistCDF(gA, gB, x)-gAlpha;
end;

'Find the inverse of the complement of the cumulative F Distribution
'alpha The probability level
'a,b, The degrees of freedom of the two distributions
Func InvFCDFComp(alpha, a, b)
var x;
gA := a; gB := b; gAlpha := alpha;
ZeroFind(x, ZeroFRoot, 0, 20);
return x;
end;
```

F-Test for equality of two standard deviations

The simplest use of the F-test is to determine if the standard deviation of two populations are the same. See this NIST handbook section for an explanation. The `FTestSameSd()` function calculates the one-tailed probability value and also returns the critical F statistic values for both the one-tailed test (α) and also for a two-tailed test ($\alpha/2$). For a two-tailed test, both the critical levels at $\alpha/2$ and $1-\alpha/2$ apply. For the one-tailed test, only one of the levels at α or $1-\alpha$ apply. In this case, the F statistic is just the ratio of the squares of the two standard deviations and the number of degrees of freedom is the number of samples of each distribution minus 1:

```
'F-test for equal standard deviations
'n1%  Number of items in sample 1
'sd1  Standard deviation of sample 1
'n2%  Number of items in sample 2
'sd2  Standard deviation of sample 2
'alpha The probability level to calculate FCrit
Func FTestSameSd(n1%, sd1, n2%, sd2, alpha)
var f := sd1/sd2;      'The test statistic is the ratio...
f *= f;                '...of the two sd's squared
var prob := OneMinusFDistCDF(n1%-1, n2%-1, f);
PrintLog("\nF-Test for same standard deviations\n");
PrintLog("Number of samples    = %8d %8d\n", n1%, n2%);
PrintLog("Standard deviations = %8g %8g\n", sd1, sd2);
PrintLog("Test statistic (F)    = %8g\n", f);
PrintLog("Prob (one-tailed)    = %8g\n", prob);
PrintLog("Upper critical level at alpha/2 = %8g\n", InvFCDFComp(alpha/2, n1%-1, n2%-1));
PrintLog("Upper critical level at alpha   = %8g\n", InvFCDFComp(alpha, n1%-1, n2%-1));
PrintLog("Lower critical level at alpha   = %8g\n", InvFCDFComp(1-alpha, n1%-1, n2%-1));
PrintLog("Lower critical level at alpha/2 = %8g\n", InvFCDFComp(1-alpha/2, n1%-1, n2%-1));
return prob;
end;
```

With the following data:

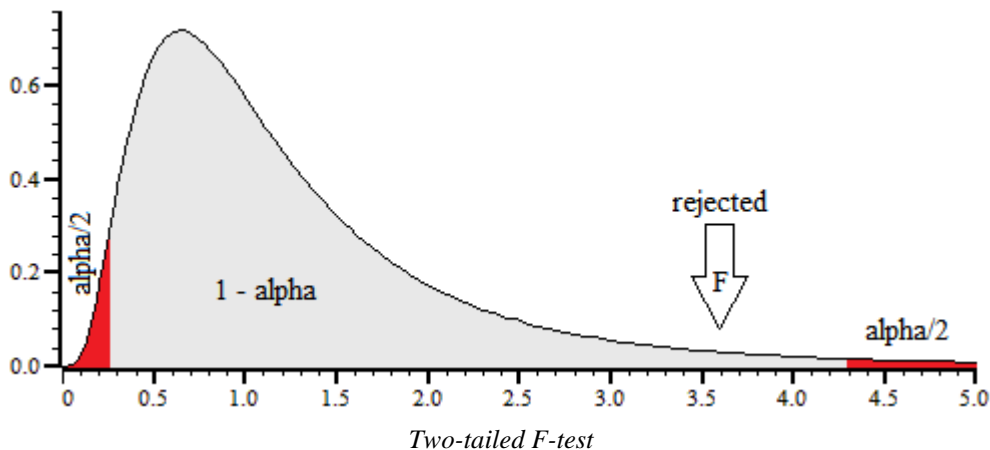
Sample	Items	sd
1	11	4.9082
2	9	2.5874

We use the script to test at an alpha level of 0.05:

```
FTestSameSd(11, 4.9082, 9, 2.5874, 0.05);
```

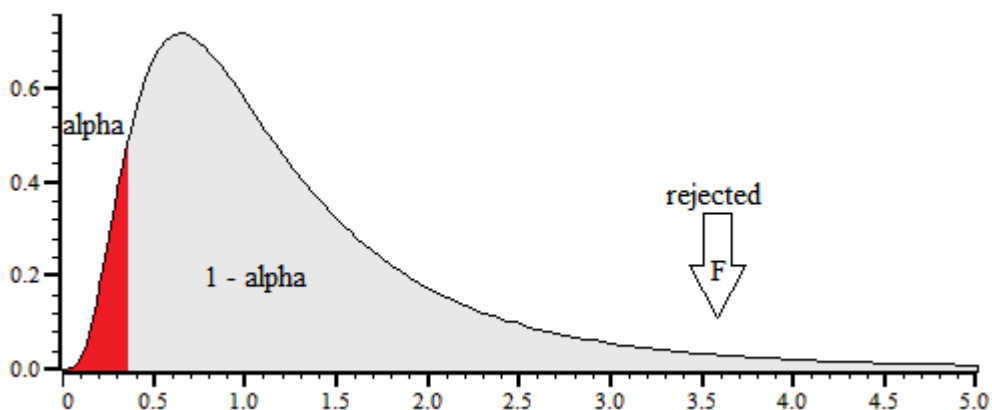
and the output is:

```
F-Test for same standard deviations
Number of samples    =      11      9
Standard deviations =  4.9082  2.5874
Test statistic (F)   =  3.59847
Prob (one-tailed)   =  0.0411439
Upper critical level at alpha/2 =  4.29513 (two-tailed test upper limit)
Upper critical level at alpha   =  3.34716 (sd1 > sd2 one-tailed limit)
Lower critical level at alpha   =  0.325557 (sd1 < sd2 one-tailed limit)
Lower critical level at alpha/2 =  0.259411 (two-tailed test lower limit)
```



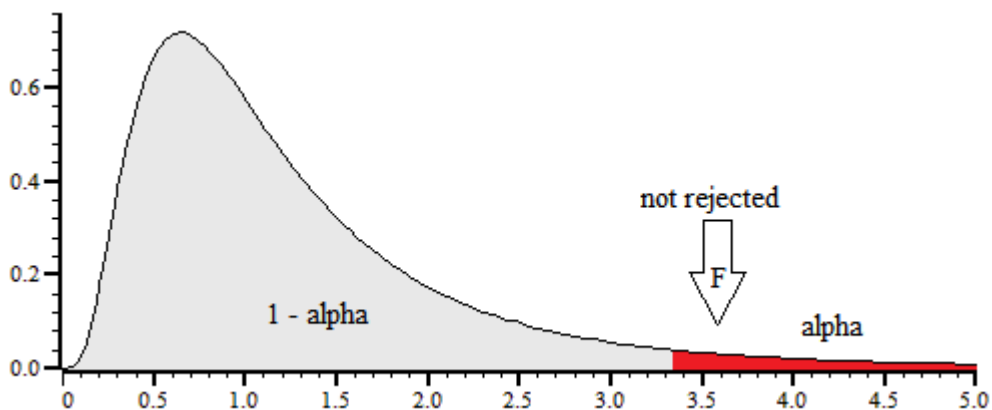
Given this result, for a two-tailed test the hypothesis that the standard deviations are different is rejected at the

5% level as the F statistic does not fall outside the range 0.259411 to 4.29513. The probability value returned (that the result occurred by chance) is for comparison with alpha for a one-tailed test; for a two-tailed test compare it with $\alpha/2$ or double it for comparison with alpha .



One-tailed F test for less

For a one-tailed test that the standard deviation is less, the result is a clear rejection (as you would expect as the value was greater).



One-tailed F-test for greater

For a one-tailed test, the probability value is less than alpha, and we can see that the hypothesis that sample 1 standard deviation is greater than sample 2 is not rejected at the 5% level.

See also:

Beta(), Binomial Distribution, Student's T Distribution

BinError()

This function is used in a result view to access the optional error (standard deviation about the mean) and bin counts (number of items added to each bin) arrays. Errors are enabled for a result view created with `SetAverage()` or `SetResult()` with 4 added to `flags%`; enable bin counts by adding 32 to `flags%`. To set a non-zero error the sweep count must be greater than 1. Use `Sweeps()` to set the count. The first command variant transfers data for a single bin, the others transfer an array of bins:

```
Func BinError(chan%, bin%, newSD{, newCt%});
Func BinError(chan%, bin%, sd[]{, set%{, Ct%[]});
Func BinError(chan%, bin%, 0, set%, Ct%[]);
```

`chan%` The channel number in the result view.

`bin%` The first bin number for which to get or set the error information.

`newSD` If present, this sets the standard deviation for bin `bin%`.

`newCt%` If present, this sets the item count (if item counts are enabled) for bin `bin%`.

`sd[]` An array for standard deviations or 0 for no transfer. Transfers start at bin `bin%` in the result view. If

the array is too long, extra bins are ignored. If there is no error information, setting has no effect and reading fills `sd[]` with 0.

`set%` If present and non-zero, the values in `sd[]` and `Ct%[]` are copied to the result view. If omitted or zero, values are copied from the view into `sd[]` and `Ct%[]`.

`Ct%[]` If present, used to transfer item count information. If there is no item count, setting has no effect and reading fills the array with the current sweep count.

Returns The first command variant returns the standard deviation at the time of the call. The other variants returns the number of bins copied. If there are 0 or 1 sweeps of data the standard deviation set will be zero.

To illustrate how errors are calculated, we will assume that we are dealing with an average that is set to display the mean of the data in each bin. In terms of the script language, if the array `s[]` holds the contribution of each sweep to a particular bin, the mean, standard deviation and standard error of the mean are calculated as follows:

```
var mean, sd:=0, i%, diff, sem;
for i%:= 0 to Sweeps()-1 do mean += s[i%] next; 'form sum
mean /= Sweeps(); 'form mean data value
for i%:= 0 to Sweeps()-1 do
  diff := s[i%]-mean; 'difference from mean
  sd += diff*diff; 'sum squares of differences
  next;
sd := Sqrt(sd/(Sweeps()-1)); 'the standard deviation
sem := sd/Sqrt(Sweeps()); 'the standard error of the mean
```

We divide by `Sweeps()-1` to form the standard deviation because we have lost one degree of freedom due to calculating the mean from the data. If the result view has item counts per bin enabled (add 32 to `flags%` in `SetAverage()` or `SetResult()`) then replace `Sweeps()` by the item count for the bin.

See also:

`BinSize()`, `BinToX()`, `SetAverage()`, `SetResult()`, `Sweeps()`

BinomialC

This function calculates the Binomial coefficient nCk , which is the number of different ways to choose k items from n , which is $n!/(k!(n-k)!)$. As n factorial grows rapidly as n increases, this can be difficult to compute for even a modest value of n .

Func BinomialC(n%, k%)

`n%` The number of items from which to choose. This must be greater than 0.

`k%` The number to choose, in the range 0 to `n%`.

Returns The binomial coefficient. The return is integral, but is returned as a real value as it can be very large, for example `BinomialC(34,17)` is 23336062200, which exceeds 32-bit integer range. Floating point numbers can represent integers exactly up to some 15 digits, after which accuracy cannot be maintained. If `n%` exceeds 1029 the result can be infinity, which means it is greater than $1.7977e+308$. If you really need results for very large `n%`, you can get the logarithm of the result with `LnGamma(n%+1)-LnGamma(k%+1)-LnGamma(n%-k%+1)`.

See also:

`BetaI()`, `GammaP()` and `GammaQ()`, `LnGamma()`

Binsize()

In a result view, this returns the x axis increment per bin. In a time view, the value returned depends on the channel type.

Func Binsize({chan%});

`chan%` In a time view this is the channel from which to return information. If you omit the argument, the function returns the file time resolution in seconds. In a result view, `chan%` is ignored, and should be

omitted.

Returns In a time view, the sampling interval between points is returned for Waveform, WaveMark and RealWave channels or a negative number if the channel does not exist. Otherwise the underlying time resolution of the file in seconds is returned.

See also:

BinToX(), XToBin()

BinToX()

This converts bin numbers to x axis units in the current result view. If the current view is a time view, it converts the underlying Spike2 time units into time in seconds.

```
Func BinToX(bin);
```

bin A bin number in the result view. You can give a non-integer bin number without error. If you give a bin number outside the result view, the bin number is limited to the range of the result view before it is converted to an x axis value.

The x axis range of a result view is BinToX(0) to BinToX(MaxTime()). Do not confuse this range with XLow() to XHigh(), which is the visible range of the x axis in the current view.

In a time view, this is in the underlying time units. If the value is beyond the x axis range, it is limited to the x axis range. The value need not be integral, but you should note that all data items in the time view have time stamps that correspond with integral values of bin. The returned value is in seconds.

Returns It returns the equivalent x axis position.

See also:

BinSize(), MaxTime(), XHigh(), XLow(), XToBin()

BRead()

This reads data into variables and arrays from a binary file opened by FileOpen(). The function reads 32-bit integers, 64-bit IEEE real numbers and zero-terminated strings.

```
Func BRead(&arg1|arg1[]|&arg1%|arg1%[]|&arg1$|arg1$[] {,...});
```

arg Arguments may be of any type. Spike2 reads a block of memory equal in size to the combined size of the arguments and copies it into the arguments. Strings or string arrays are read a byte at a time until a zero byte is read.

Returns It returns the number of data items for which complete data was read. This will be less than the number of items in the list if end of file was reached. If an error occurred during the read, a negative code is returned.

For reasons of backwards compatibility, although integers are now 64-bit in Spike2 version 8, integer reads are treated as signed 32-bit numbers and the values read are sign extended into 64-bits. If you want to read 64-bit integers, use BReadSize().

See also:

FileOpen(), BReadSize(), BRWEndian(), BSeek(), BWrite()

BReadSize()

This converts data into variables and arrays from a binary file opened by FileOpen(). The function reads 8, 16 and 32-bit integers and converts them to 32-bit integers, and 32 and 64-bit IEEE real numbers and converts them to 64-bit reals. It also reads strings from fixed-size regions in the file (zero bytes are ignored during the read). The read is from the current file position. The current position after the read is the byte after the last byte read.

```
Func BReadSize(size%, &arg|arg[]|&arg%|arg%[]|&arg$|arg$[] {,...});
```

size% The bytes to read for each argument. Legal values depend on the argument type:

Integer	1, 2, 4, 8	Read 1, 2 or 4 bytes and sign extend to 64-bit integer or read 8 bytes (no sign extend needed).
	-1, -2, -4	Read 1, 2 or 4 bytes and zero extend to 64-bit integer.
Real	4	Read 4 bytes as 32-bit real, convert to 64-bit real.
	8	Read 8 bytes as 64-bit real.
String	n	Read n bytes into a string. Null characters end the string as seen by the script, but all n bytes are read..
	0	Read 1 character at a time until either a Null character or the end of the file is found.
	-n	Read 1 character at a time until either a Null character or n characters have been read (added at version 8.00).

arg The target variable(s) to be filled with data. **size%** applies to all targets.

Returns It returns the number of data items for which complete data was read. This will be less than the number of items in the list if end of file was reached. If an error occurred during the read, a negative code is returned.

See also:

`FileOpen()`, `BRead()`, `BRWEndian()`, `BSeek()`, `BWrite()`

BRWEndian()

This gets and sets the "endianism" of binary data files. This affects numeric data used with `BRead()`, `BReadSize()`, `BWrite()` and `BWriteSize()`. PC programs normally use little-endian data (least significant byte at lowest address). Some systems, including the Macintosh, use big-endian data (most significant byte at lowest address). Binary files are little-endian by default.

Most users do not need to use this routine. Only use it if you are writing binary files for use on a big-endian computer or reading binary files that were generated with a big-endian system.

```
Func BRWEndian({new%});
```

new% Omit or set -1 for no change. Set 0 for little-endian and 1 for big-endian.

Returns The current endianism as 0 for little, 1 for big or a negative error code.

See also:

`FileOpen()`, `BRead()`, `BReadSize()`, `BSeek()`, `BWrite()`, `BWriteSize()`

BSeek()

This function moves and reports the current position in a file opened by `FileOpen()` with a **type%** code of 9. The next binary read or write operation to the file starts from the position returned by this function.

```
Func BSeek({pos% {, rel%}});
```

pos% The new file position as the byte offset the start, the current position, or the end of the file. If **pos%** is omitted, the position is not changed and the function returns the current position.

rel% This determines to what the new position is relative:

- 0 Relative to the start of the file (same as omitting the argument).
- 1 Relative to the current position in the file.
- 2 Relative to the end of the file.

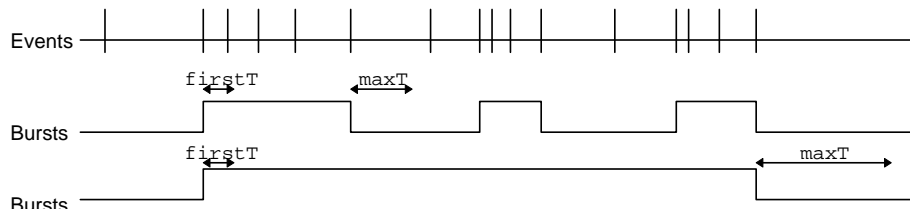
Returns The new file position relative to the start of the file or a negative error code.

See also:

`FileOpen()`, `BReadSize()`, `BRead()`

BurstMake()

This function extracts burst start and end times from an event channel and writes them to a memory channel. Three parameters control the formation of bursts: the interval between the first two events in a burst, the interval between the last event in a burst and any following event, and the minimum events in a burst.



The diagram illustrates the method of forming bursts. In the first case, with a short maximum interval between events, the algorithm finds three bursts. In the second case with a longer period, the algorithm detects one burst. The command for this function is:

```
Func BurstMake(mChan%, eChan%, sTime, eTime, maxT{, firstT{, minE%}});
```

mChan% The channel number of an event, level event or marker channel created by MemChan() for the output. If this is a marker channel, the start of each burst is given a first marker code of 00 and the end has a first marker code of 01.

eChan% The channel number of an event channel to search for bursts.

sTime The start of the time range to search for bursts.

eTime The end of the time range to search for bursts.

maxT The maximum time between two events (after the first pair) for the events to lie in the same burst.

firstT The maximum time between the first two events in a burst. If omitted, maxT sets the time interval for the first pair of events.

minE% The minimum number of events that can make up a burst. The default is 2.

Returns The function returns the number of bursts found, or a negative error code.

See also:

BurstRevise(), BurstStats(), MemChan()

BurstRevise()

This modifies a list of times, indicating bursts and produces a new list of bursts that have inter-burst intervals and burst durations greater than specified minimum times (See BurstStats() for more information).

```
Func BurstRevise(mChan%, eChan%, sTime, eTime, minI, minD);
```

mChan% A memory channel created by MemChan() holding event data to which the output of this command is added. This can be the same channel as eChan%, but if it is, the output is generated by deleting unwanted events from the channel.

eChan% The event or marker channel to read burst times from.

sTime The start of the time range to revise.

eTime The end of the time range to revise.

minI The minimum interval between the end of one burst and the start of the next. Bursts with shorter intervals are amalgamated.

minD The minimum duration of a burst. Shorter bursts are deleted.

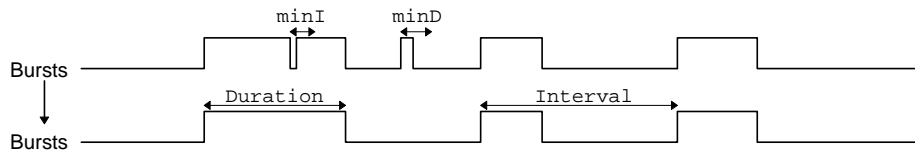
Returns The number of bursts in the output or a negative error code.

See also:

BurstMake(), BurstStats(), MemChan()

BurstStats()

This command returns statistics on bursts (possibly made by `BurstMake()`). Additional rules can be applied to the bursts before the statistics are calculated to amalgamate bursts that are too close together and to delete bursts that are too short. The statistics are the mean and standard deviation of the duration of the bursts and the intervals between the start of one burst and the start of the next.



```
Func BurstStats(eChan%, sTime, eTime, &meanI, &sdI, &meanD,
&sdD{,minI{,minD}});
```

`eChan%` The event channel containing burst data. The first event found in the time range is assumed to be the start of a burst, the second the end, and so on.

`sTime` The start of the time range to process.

`eTime` The end of the time range to process.

`meanI` This is a real variable that is returned holding the mean interval between the starts of bursts (as long as at least 2 bursts were found).

`sdI` This is a real variable that is returned holding the standard deviation of the mean interval (as long as at least 3 bursts were found).

`meanD` This is a real variable that is returned holding the mean burst duration (as long as at least 1 burst was found).

`sdD` This is a real variable that is returned holding the standard deviation of the burst durations (as long as at least 2 bursts were found).

`minI` This optional value sets the minimum interval between the end of one burst and the start of the next. It takes the value 0.0 if omitted. Bursts that are closer than this are amalgamated for the purpose of forming statistics.

`minD` This optional value (taken as 0.0 if omitted) sets the minimum burst duration. Bursts shorter than this (after amalgamations) are ignored in the statistics.

Returns It returns the number of bursts used for the statistics, or a negative error code.

This command is used in scripts that optimise the values of `minI` and `minD` for forming bursts. For example, when bursting is known to follow a cyclical pattern, one would like to find values of `minI` and `minD` that minimise the total coefficient of variance:

$$\text{Total coefficient of variance} = (\text{sdD}/\text{meanD})^2 + (\text{sdI}/\text{meanI})^2$$

Once suitable values are found, the `BurstRevise()` command can generate a memory channel holding bursts based on the optimised parameters.

See also:

`BurstMake()`, `BurstRevise()`, `MemChan()`

BWrite()

This function writes binary data values and arrays into a file opened by `FileOpen()` with a `type%` code of 9. The function writes 32-bit integers (for backwards compatibility with version 7), 64-bit IEEE real numbers and strings. The output is at the current position in the file. The current position after the write is the byte after the last byte written. To write 64-bit integers see `BWriteSize()`.

```
Func BWrite(arg1 {,arg2 {,...}});
```

`arg` Arguments may be of any type, including arrays. Spike2 fills a block of memory equal in size to the combined size of the arguments with the data held in the arguments and copies it to the file. An

integer uses 4 bytes and a real uses 8 bytes. A string is written as the bytes in the string plus an extra zero byte to mark the end. Use `BWriteSize()` to write a fixed number of bytes.

Returns It returns the number of arguments for which complete data was written. If an error occurred during the write, a negative code is returned.

The following example writes data from a channel of a data file as binary:

```
'Declare variables:
var ok%,chan%,fh%,bh%,close%:=0, fmt%:=0, fmt${3};
fmt${0} := "Integer"; fmt${1}:="32-bit IEEE real";
fmt${2} := "64-bit IEEE real";
if ViewKind(<>0 then
    fh% := FileOpen("", 0, 0, "File to dump as binary");
    if fh%<=0 then Message("No file to dump"); halt endif;
    close% := 1;                'say we should close file
endif;
fh% := View();                'get the starting view handle
DlgCreate("Binary file dump"); 'Start new dialog
DlgChan(1,"Choose the channel to dump",16511);
DlgList(2,"Format for the output",fmt${});
ok% := DlgShow(chan%, fmt%);  'ok is 0 if user cancels
if ok% <> 0 then                'dump if user selects a channel
    bh% := FileOpen("", 9, 1, "File to dump channel to");
    if bh% > 0 then BinDump(bh%, fh%, chan%, fmt%) endif;
endif;
View(bh%);FileClose();        'close the binary file
if close% then                  'if we opened the file..
    View(fh%);                  '...the move to it...
    FileClose();                '...and close it again
endif;

Proc BinDump(bh%, fh%, ch%, fmt%)    'binary file, data file, channel
docase
    case fmt% = 0 then IntDump(bh%, fh%, ch%);    'Integer format
    case fmt% = 1 then RealDump(bh%, fh%, ch%, 4); '4 byte real
    case fmt% = 2 then RealDump(bh%, fh%, ch%, 8); '8 byte real
endcase;
end;

Proc RealDump(bh%, fh%, ch%, bytes%)
const BSZ% := 8000;                'buffer size
var work[BSZ%];                    'work space
var t := 0, n%, kind%;              'start time, items read
View(fh%);                          'in data file view
kind% := ChanKind(ch%);
if (kind%=1) or (kind%=8) then      'waveform channel
    repeat
        n% := ChanData(ch%, work[], t, MaxTime(), t);
        if n% > 0 then                'if we got data, then
            View(bh%).BWriteSize(bytes%, work[:n%]); 'Output it
            t := t + n% * BinSize(ch%); 'time of next point
        endif;
    until n% <= 0;                    'until no points left
else
    repeat
        n% := ChanData(ch%, work[], t, MaxTime());
        if n% > 0 then                'if we got data then...
            View(bh%).BWriteSize(bytes%, work[:n%]); 'Write it
            t := work[n%-1]+BinSize(ch%); 'next search start
        endif;
    until n% <= BSZ%;                'until buffer not full
endif;
end;

proc IntDump(bh%, fh%, ch%)
const BSZ% := 8000;
var work[BSZ%];                    'work space
var t := 0, n%, kind%;              'start time, items read
View(fh%);                          'in data file view
kind% := ChanKind(ch%);
if (kind%=1) or (kind%=8) then      'waveform channel
    repeat
        n% := ChanData(ch%, work%[], t, MaxTime(), t);
```

```

    if n% > 0 then          'if we got data
        View(bh%).BWriteSize(2, work%[:n%]); 'Output it
        t := t + n% * BinSize(ch%); 'time of next point
    endif;
    until n% <= 0;
else
    'Times as 64-bit integer
    repeat                '(needs version 8 or later)
        n% := ChanData(ch%, work%[], t, MaxTime());
        if n% > 0 then    'if we got any data
            View(bh%).BWriteSize(8, work%[:n%]); 'Write it
            t := BinToX(work%[n%-1]+1); 'next search start
        endif;
    until n% <= BSZ%;
endif;
end;

```

See also:

FileOpen(), BRead(), BReadSize(), BRWEndian(), BWriteSize()

BWriteSize()

This function writes variables or arrays as binary into a file opened by FileOpen() with a type% code of 9. It writes 8, 16, 32 and 64-bit integers and 32 and 64-bit reals and strings. It allows you to write formats other than the 64-bit integer and 64-bit real used internally by the script and to write variable-length strings into fixed-size fields in a binary file. Support for 64-bit integers was added at version 8.

```
Func BWriteSize(size%, arg1 {,arg2 {,...}});
```

size% Bytes to write for each argument (or array element if the argument is an array). Legal values of size% depend on the argument type:

Integer	1, 2, 4	Write least significant 1, 2 or 4 bytes.
	8	Write all 8 bytes of the integer.
Real	4	Convert to 32-bit real and write 4 bytes.
	8	Write 8 bytes as 64-bit real.
String	n	Write n bytes. Pad with zeros if the string is too short.

arg The target variable(s) to be filled with data. size% applies to all targets.

Returns It returns the number of data items for which complete data was written or a negative error code.

See also:

FileOpen(), BRead(), BReadSize(), BRWEndian(), BWrite()

C

Ceil()

Returns the next higher integral number of the real number or array. Ceil(4.7) is 5.0, Ceil(4) is 4. Ceil(-4.7) is -4.

```
Func Ceil(x|x[]{|...});
```

x A real number or a real array.

Returns 0 or a negative error code for an array or the next higher integral value.

See also:

Abs(), ATan(), Cos(), Exp(), Floor(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan(), Trunc()

Chan...()

Chan\$()

This function converts a channel number or a list of channel numbers into a string. Memory, duplicate and virtual channels are listed as they appear on screen (not just as numbers). If a channel does not exist in the current view, it is represented as a number.

```
Func Chan$(chan%|chan%[ ]);
```

chan% Either a channel number or an array of integers in the same format as a channel specifier (the first element holds the number of items, followed by the channel numbers).

Returns A channel specification string, for example "1,3,5..8,m2,v1a".

See also:

ChanList()

ChanCalibrate()

This function is equivalent to the Analysis menu Calibrate command. It changes the scale and offset of Waveform and WaveMark channels and rewrites RealWave channels so that user-defined data sections have user-defined values. The command is:

```
Func ChanCalibrate(cSpC,mode%,cfg%,t1,t2,units$,v1{,v2{,t3,t4}});
```

cSpC A channel specifier for the channels to calibrate or 1 for all, -2 for visible and -3 for selected channels. You cannot calibrate processed channels if the process changes the scale or offset or any processed RealWave channel.

mode% The calibration mode. The items in brackets are the required optional arguments.

- 0 Mean levels of two time ranges (v2, t3, t4).
- 1 Values at two time points (v2).
- 2 Set offset from mean of time range.
- 3 Set scale from mean of time range.
- 4 Square wave, upper and lower level (v2).
- 5 Square wave, amplitude (Size) only.
- 6 Peak to peak amplitude and mean (v2).
- 7 RMS amplitude about mean (v2).
- 8 Area under curve, assume zero at end.
- 9 Areas under curve, two time ranges (v2, t3, t4).
- 10 Slope of best-fit line to the data, the offset not changed.

cfg% If non-zero, the new calibration is saved in the sampling configuration.

t1,t2 In all modes except mode 1, these are the start and end of the first time range. In mode 1 these times correspond to the two calibration values v1 and v2.

units\$ The units to apply to the channel.

v1,v2 The values in user units that correspond to the times or time ranges. v1 is always used; v2 is only used in modes 0, 1, 4, 6, 7 and 9.

t3,t4 A second time range. These values are used in modes 0 and 9.

Returns The return value is the sum of the following values:

- 1 A channel in the list did not exist or was the wrong type.
- 2 A channel was processed OK.

- 4 Unknown or unimplemented calibration mode.
- 8 A time range had the end past the start or two time ranges overlapped or the two times in mode 1 were the same.
- 16 The v1 and/or v2 values were too big, too small or too similar.
- 32 Not enough data to process at least one channel in the list.
- 64 The data is unsuitable. For example, in mode 0 mean levels must differ by at least the standard deviation of the data around the mean.

See also:

Channel specifiers, ChanOffset(), ChanScale(), ChanUnits\$()

ChanColour()

Deprecated. This returns and optionally sets the colour of a channel in a time or result view to a palette colour. This colour overrides the application colour set for the drawing mode of the channel. This function is provided for backwards compatibility and should only be used when a script must work with older versions of Spike2. Use ChanColourGet() and ChanColourSet() instead to set colours as RGB combinations.

```
Func ChanColour(chan%, item%{, col%});
```

chan% A channel in the time or result view.

item% The colour item to get or set; 0=background, 1=primary, 2=secondary colour.

col% If present, the new colour index for the item. There are 40 colours in the palette, indexed 0 to 39. Use -1 to revert to the application colour for the drawing mode.

Returns The palette colour index that is nearest to the item colour at the time of the call, -1 if no colour is set or a negative error code if the channel does not exist. **Beware:** if you set a colour index for an item and read it back you may not get the index you set if another index holds the same colour.

See also:

Colour dialog, ChanColourGet(), ChanColourSet(), Colour(), PaletteGet(), PaletteSet(), ViewColour(), ViewUseColour(), XYColour()

ChanColourGet()

Returns a channel item RGB colour in a time, result or XY view. This was added at Spike2 version 7.07.

```
Func ChanColourGet(chan%, item%{, &r, &g, &b});
```

chan% A channel in the time, result or XY view.

item% The colour item to get; 0=background, 1=primary, 2=secondary colour for time and result views, the fill colour for an XY channel.

r g b If present, these variables are returned holding the colour as red, green and blue values in the range 0 to 1.0.

Returns 1 if the channel colour is overridden, 0 if not, negative for no channel.

See also:

Colour dialog, ChanColourSet(), ColourGet(), ViewColourGet()

ChanColourSet()

Sets a channel item RGB colour in a time, result or XY view. A change to a colour item that would make visible difference will cause the affected view to become invalid and it will repaint at the next opportunity. This was added at Spike2 version 7.07.

```
Func ChanColourSet(chan%, item%{, r, g, b});
```

chan% A channel in the time, result or XY view.

item% The colour item to get; 0=background, 1=primary, 2=secondary colour for time and result views, the fill colour for an XY channel.

r g b If present, these arguments set the colour as red, green and blue values in the range 0 to 1.0.

Returns 0 or a negative error code if the channel does not exist.

See also:

Colour dialog, ChanColourGet(), ColourSet(), ViewColourSet()

ChanComment\$()

This returns or sets the comment string for a channel in a time or result view. The comment can be up to 71 characters long in a time view, and is unlimited in a result view (but is truncated to 79 characters if a result view is saved to a file). It is an error to use this in any other view type.

```
Func ChanComment$(chan%{, new$});
```

chan% A channel in the time or result view. In a result view, if the channel is a duplicate, the comment is returned from or set in the source channel.

new\$ An optional string with a new comment. If the string is too long, it is truncated.

Returns It returns the comment string for the designated channel. If the channel does not exist, the function does nothing and returns an empty string.

See also:

ChanTitle\$(), FileComment\$()

ChanData()

Fills an array with waveform data from a waveform or RealWave channel or with event times from all other channel types. Use NextTime() and LastTime() to get data attached to WaveMark, RealMark and TextMark channels and marker codes. If you want the number of events between two times you should use Count(). You can also use ChanMeasure() to calculate commonly required values between two times.

```
Func ChanData(chan%, arr[]|arr%[], sTime, eTime {,&fTime});
```

chan% The data channel in the current time view to read data from.

arr A real or integer array. Real arrays collect waveform data in user units and event times in seconds. Integer arrays collect waveforms in ADC units and event times in the underlying time units (as returned by Binsize()). RealWave data is converted to integers using the channel scale and offset.

sTime The first data value returned is at or after this time in seconds.

eTime The last data point returned is before or at this time in seconds.

fTime This optional argument is a variable that is set to the time of the first data point.

Returns The number of data values placed in the array. Only contiguous waveform data is returned; gaps terminate a read.

Using ChanData()

You will normally use `ChanData()` iteratively to work your way through a channel of data. You can find examples of this in the `BWrite()`, `Count()` and `ChanNew()` command descriptions and for several other commands if you search this help for `ChanData`. Allocating a fixed size array and reading into it once (assuming that this will always be big enough) is probably a mistake. The example for `BWrite()` is particularly helpful as it shows how to iterate through waveform and event data collecting the data as either real or integer numbers.

Event times

Event times read as integers from 64-bit `smrx` files are 64-bit values. If you are migrating from earlier versions of Spike2 this should not cause you any problems. However, you should bear this in mind when writing scripts that must also run on earlier versions of Spike2 where all times are limited to 32-bit integer values.

See also:

`Binsize()`, `ChanMeasure()`, `ChanScale()`, `ChanValue()`, `ChanWriteWave()`, `Count()`, `NextTime()`

ChanDelete()

This deletes a channel from a time, result or XY view. You can make the user confirm time view channel deletion if the channel is stored in the file. Duplicated channels, and memory buffer channels are not confirmed. In a result view, you may only delete duplicate channels. In an XY view you cannot delete the last XY channel as XY views must always have at least one channel. Channels are always numbered consecutively in an XY view, so if you delete a channel, the channel numbers of any higher numbered channels will change.

```
Func ChanDelete(cSpc {,query%});
```

`cSpc` A channel specifier for the channels to delete or -1 for all, -2 for visible or -3 for selected. In an XY view only channel numbers greater than 0 are allowed.

`query%` If present and non-zero, the user is asked to confirm the deletion if the channel is part of a time view. You cannot delete channels that are being sampled.

Returns 0 if the channel was deleted or a negative error code if the user cancelled the operation or tried to delete the last XY channel or for other problems.

See also:

Channel specifiers, `ChanDuplicate()`, `MemChan()`, `XYDelete()`, `XYSetChan()`

ChanDuplicate()

This duplicates a time or result view channel. It can also be used in the Edit WaveMark view to delete all duplicates of the current channel and generate duplicates with suitable marker filter setting for each template. Use `DupChan()` to find duplicates of a channel.

```
Func ChanDuplicate({chan%});
```

`chan%` For time or result views, this is channel number to duplicate; this channel must exist. This argument must be omitted for Edit WaveMark views.

Returns For time or result views it returns the channel number of the duplicate. For an Edit WaveMark view, it returns the number of duplicates that were created. Any error stops the script.

For time and result views the new channel is not displayed. Use `ChanShow()` to make it visible. In an Edit WaveMark view, channels are made visible immediately. The following example duplicates a time or result view channel and makes it visible:

```
var ch%;ch% := ChanDuplicate(1); 'create a duplicate
ChanShow(ch%); 'make visible
```

See also:

`ChanShow()`, `ChanDelete()`, `DupChan()`, `MemChan()`

ChanFit()

This function together with `ChanFitCoef()` and `ChanFitShow()` incorporates the functionality of the Analysis menu Fit Data dialog. `ChanFit()` has three variants that: initialise ready for a new fit, perform the fit and return information about the last fit. The current window must be a time, result or XY view to use these functions.

Initialise fit information

This command associates a fit with a channel. The fit parameters and the coefficient limits are reset to their default values, the coefficient hold flags are cleared and any existing fit for this channel is removed.

```
Func ChanFit(chan%, type%, order%);
```

`chan%` The channel number to work on. Each channel in a time, result or XY view can have one fit associated with it.

`type%` 0=Clear fit, 1=Exponential, 2=Polynomial, 3=Gaussian, 4=Sine, 5=Sigmoid.

`order%` The order of the fit. This is 1 or 2 for an exponential or Gaussian fit, 1 for a Sine or Sigmoid fit and 1 to 5 for a polynomial fit. If `type%` is 0 this should also be 0.

Returns 0 if the command succeeded.

Perform the fit

This variant of the command does the fit set by the previous variant.

```
Func ChanFit(chan%, opt%, start|start$, end|end${, ref|ref${, &err{, maxI%{, &iTer%{, covar[[[]]]}}}});
```

`chan%` A channel number in the current view that has had a fit initialised.

`opt%` This is the sum of:

- 1 Estimate the coefficients before fitting, else use current values.
- 2 Display the fitted data. Use `ChanFitShow()` to change the displayed range.

`start` This is the start of the fit range as a value or as a dialog expression string that is to be evaluated.

`end` The end of the fit range in x axis units as a value or a string to evaluate.

`ref` The reference time as a value or a string to evaluate. If omitted start is used.

`err` If present, this optional variable is updated with the chi-squared or least-squares error between the fit and the data.

`maxI%` If present, this sets the maximum number of iterations. If omitted, the current number set for the channel is used. The system default number is 100.

`iTer%` If present, this integer variable is updated with the count of iterations done.

`covar` An optional two dimensional array of size at least `[nCoef][nCoef]` that is returned holding the covariance matrix when the fit is complete. It is changed if the return value is -1, 0 or 1. However, the values it contains are probably not useful unless the return value is 0.

Returns 0 if the fit is complete, 1 if max iterations done, or a negative error code: -1=the fit is not making progress (results may be OK), -2=the fit failed due to a singular matrix, -5=the fit caused a floating point error, -6=too little data for the number of coefficients, -7=unknown fitting function, -8=ran out of memory during the fit (too many data points), -9=the fit was set up with bad parameters.

Get fit information

This variant of the command returns information about the current fit set for a channel.

```
Func ChanFit(chan%{, opt%});
```

`chan%` The channel number of the fit to return information about.

`opt%` This determines what information to return. If omitted, the default value is 0. Positive values return

information about the fit that is set-up to be done next. Negative values return information about the last fit that was done and that can be displayed. The returned information for each value of `opt%` is:

<code>opt%</code> Returns	<code>opt%</code> Returns
0 Fit type of next fit	1 Fit order of next fit
-1 1=a fit exists, 0=no fit exists	-9 User-defined x draw start
-2 Type of existing fit or 0	-10 User-defined x draw end
-3 Order of existing fit	-11 1=chi-square, 0=least-square
-4 Chi or least-squares error	-12 Last fit result code
-5 Fit probability (estimated)	-13 Number of fitted points
-6 X axis value at fit start	-14 Number of fit iterations used
-7 X axis value at fit end	-15 R-square value
-8 Reference x value	-16 Adjusted R-square value

Returns The information requested by the `opt%` argument or 0 if `opt%` is out of range.

See also:

Fit Data dialog, More about fitting, Dialog expressions, `ChanFitCoef()`, `ChanFitShow()`, `ChanFitValue()`, `FitExp()`, `FitPoly()`

ChanFitCoef()

This command gives you access to the fit coefficients for a channel in the current time, result or XY view. You can return the values from any type of fit and set the initial values and limits and fix values for iterative fits. There are two command variants:

Set and get coefficients

This command variant lets you read back the current coefficient values and set the coefficient values and limits for iterative fitting:

```
Func ChanFitCoef(chan%, num%, new{, lower{, upper}}});
```

`chan%` The channel number of the fit to access.

`num%` If omitted, the return value is the number of coefficients in the current fit. If present, it is a coefficient number. The first coefficient is number 0 and the coefficients are numbered to match their order in the Fit Data dialog. With `num%` present, the return value is the coefficient value of the current fit, or if there is no fit, the coefficient value that would be used as the starting point for the next iterative fit.

`new` If present, this sets the value of coefficient `num%` for the next iterative fit on this channel.

`lower` If present, this sets the lower limit for coefficient `num%` for the next iterative fit on this channel. There is currently no way to read back the coefficient limits. There is no check that the limits are valid.

`upper` If present, this sets the upper limit for coefficient `num%` for the next iterative fit on this channel.

Returns The number of coefficients or the value of coefficient `num%`.

Get and set the hold flags

This variant sets the hold flags (equivalent to the Hold check boxes in the Fit Data dialog Coefficients tab).

```
Func ChanFitCoef(chan%, hold%[]);
```

`chan%` The channel number of the fit to access.

`hold%` An array of integers to correspond with the coefficients. If the array is too long, extra elements are ignored. If it is too short, extra coefficients are not affected. Set `hold%[i%]` to 1 to hold coefficient `i%` and to 0 to fit it. If `hold%[i%]` is less than 0, the hold state is not changed, but `hold%[i%]` is set to 1 if the corresponding coefficient is held and to 0 if it is not held.

Returns This always returns 0.

See also:

`ChanFit()`, `ChanFitShow()`, `ChanFitValue()`, `FitExp()`, `FitPoly()`

ChanFitShow()

This controls the display of data fitted to a channel in the current time, result or XY view.

```
Func ChanFitShow(chan%{, opt%{, start|start${, end|end$}}});
```

chan% The channel number of the fit to access.

opt% If present and positive, this is the sum of:

- 1 Display the fitted data (if not added, any fit on the channel is hidden).
- 2 Use the user-defined display range rather than the fitting range.

If **opt%** is omitted or positive, the return value is the current option value. Use negative values to return the user-defined display range: -1=return the start, -2=return the end.

start If present, this is an x axis value or a string holding a dialog expression to be interpreted as an x axis value that sets the start of the user-defined display range.

end If present, it sets the end of the user-defined display range as **start** sets the start. The allowed range is limited to 4 times the width of the fitted data before and after the fitted data.

Returns The current **opt%** value or the information requested by **opt%**. If there is no fit defined for the channel, the return value is 0.

See also:

Dialog expressions, ChanFit(), ChanFitValue(), ChanFitCoef(), FitExp(), FitPoly()

ChanFitValue()

This function returns the value at a particular x axis value of the fitted function to a channel in the current time, result or XY view.

```
Func ChanFitValue(chan%, x);
```

chan% The channel number of the fit to access.

x The x axis value at which to evaluate the current fit. Some fitting functions can overflow floating point range if **x** is outside the fitted range of the function.

Returns The value of the fitted function at **x**. If the result is out of floating point range, the function may return a floating point infinity or a NaN (Not a Number) value or a 0. If there is no fit, the result is 0.

See also:

ChanFit(), ChanFitShow(), ChanFitCoef(), FitExp(), FitPoly()

ChanHide()

Hide a channel, or a list of channels. Hiding a channel that doesn't exist has no effect.

```
Proc ChanHide(cSpc {, cSpc...});
```

cSpc A channel specifier or -1 for all, -2 for visible, and -3 for selected channels.

See also:

Channel specifiers, ChanShow(), ChanList(), ChanVisible(), View menu show/hide

ChanImage()

You can set a background image behind each channel in a Time, Result or XY view. Images are read from .bmp, .jpg, .jpeg, .png, .tif or .tiff files on disk. See the View menu Channel Image command for details. There are three command variants:

```

Func ChanImage(chan%, path$);
Func ChanImage(chan%, mode%, opac{, x1, y1, xh, yh});
Func ChanImage(chan%, get%{, &path$});

```

- chan%** A channel number in the current view. In an XY view, all channels share the same bitmap.
- path\$** The file name to the image. The first command variant sets the image, the third variant reads back the file name. Setting an empty name releases any memory used to cache the bitmap within the program. Setting an image does not change the display mode; use the second variant to make sure that the image is visible. Set the path to "<CB>" to use whatever image happens to be on the clipboard.
- mode%** There are three display modes that you can set in the second command variant: 0=no display, 1=fill background, 2=fill rectangle. You can also set mode -1, meaning no mode change.
- opac** You can control the image opacity in the range 0.0 (transparent) to 1.0 (opaque). You can also set the value -1 for no change.
- x1-yh** When **mode%** is 2, these four arguments set the rectangle, in x and y axis units, that contains the bitmap image.
- get%** This is used in the third command variant to read back the current settings. You can set -1 to read the mode, -2 to read the opacity, and -3 to -6 to read the x1, y1, xh and yh values.
- Returns** The first variant returns 1 if a bitmap was read, 0 if it was not (either file not found, or no image was set) or a negative error code. The second returns 0, the third returns the requested information.

We do not provide any method to size a channel area so a bitmap becomes a specific size. However, you can use `ChanPixel()` to find the scaling between x and y axes and pixels. It is possible to then iteratively change the window size (or adjust the channel weight with `ChanWeight()`) to get the desired effect.

See also:

`ChanColour()`, `ChanPixel()`, `ChanWeight()`

ChanIndex()

Get or set the index associated with a channel in a time view. RealMark channels use the index to select the data value to display.

```

Func ChanIndex(chan%{, index%});

```

- chan%** A channel number. Currently, this is useful only with RealMark channels.
- index%** If present and positive, this sets the zero-based channel index. If omitted or -1, the command returns the current index. If -2, the command returns the number of index values for the channel.
- Returns** Either the channel index at the time of the call, or if **index%** is -2, the number of possible index values.

See also:

`MarkInfo()`

ChanKey()

Channel keys display addition information about a channel. Currently XY views have a key to identify the displayed channels and time view waveforms drawn in Sonogram mode have a key to identify the dB levels of the data. This command is used to control keys in a generic way; there is also the `XYKey()` command that works only with XY keys. There are four command variants:

```

Func ChanKey(chan%, type$, flags%, x, y{, s{, w{, h}}});
Func ChanKey(chan%, 0);
Func ChanKey(chan%, set%, val|val$);
Func ChanKey(chan%, get%, &get$);

```

The first variant displays a key or modifies an existing key and returns the previous **flags%** value. The second

variant hides the key, but does not change any other settings, and returns the previous `flags%` value. The third variant sets key-specific values and the last variant reads back key-specific values.

`chan%` A channel number in the current view. In an XY view this is ignored and should be set to 0.

`type$` The type of the key to create. This is case insensitive and we currently support "Sono" and "XY".

`flags%` The sum of a set of values that control the appearance (or non-appearance) of a key. If you set `flags%` to 0 you can omit the remaining arguments to hide the key. The values are:

Value	Meaning
1	Make the key visible. If this value is not added, the key is hidden.
2	Controls the key background. If set the key is transparent (XY) or uses the colour of the lowest intensity for the sonogram. If not set, the key uses the channel background colour.
4	If set, the key has a border. If unset, there is no border.
8	Sets the key orientation (if applicable). If set, the key is vertical, otherwise it is horizontal. This is only applicable to sonogram keys.
16	If set, the x,y position is relative to the entire view. If clear, it is relative to the channel.

`x,y` The key position as a percentage of the available space (space less the key size). The x space runs from left to right, and is the space used by the x axis. The y space runs from top to bottom and if `flags%` has 16 added it is the entire view height, otherwise it is the channel height. The scaling is arranged so that 0,0 has the key at the top left and 100,100 has the key at the bottom right.

`s` The scale factor for the key. If omitted, a default value is used. This is currently used for the sonogram to scale the font size used in the key from the font size set for the view. You can request any size you like, but there is a limit on the maximum and minimum font sizes that will be used. We reserve negative scale values for setting absolute sizes (in the future).

`w,h` These are not currently used. They are reserved for future channel keys where the size is not calculated from the key content.

`set%` Some keys have settable parameters. This field identifies the parameter to set and `val` is the value. The parameters are numbered from 1 upwards. You can only set a value when the key is visible. Setting a value that the key does not recognise has no effect. Currently defined values are:

Key	#	Use
Sonogram	1	Sets the y axis value that corresponds to 0 dB in the sonogram key. If you set 0 or a negative value, the 0 dB point is set to the default, the equivalent of a 1 bit change of a Waveform channel. For reference, 1 bit is: $\text{Abs}(\text{ChanScale}(\text{chan\%})/6553.6)$ we need the $\text{Abs}()$ in case the scale is negative; the offset value is not relevant as a 1 bit change is a difference of two values and any offset is cancelled out.

`val` A value to set as determined by `get%`.

`get%` This returns a value set by `ChanKey(chan%, set%, val)`. Negative values read back channel key settings. The returned value is usually 0 if there is no key defined for the channel:

<code>get%</code>	Returned value
1..n	The key-specific value set by <code>ChanKey(chan%, set%, val)</code>
0	This hides the key and returns the <code>flags%</code> value before the key was hidden
-1	Return the key type in <code>get\$</code> , returns 0 if no key, 1 if a key
-2	The current <code>flags%</code> value
-3, -4	The current key x and y position
-5	The current key scale
-6,-7	The current key width and height (if controllable), otherwise 0.

See also:

`XYKey()`, Sonograms

ChanKind()

This returns the type of a channel in the current time, result or XY view.

Func ChanKind(chan%)

chan% The channel number.

Returns A code for the channel type or -3 if this is not a time, XY or result view:

0 None/deleted	3 Event (Event+)	6 WaveMark	9 Real wave
1 Waveform	4 Level	7 RealMark	120 XY channel
2 Event (Event-)	5 Marker	8 TextMark	127 Result channel

The result and XY codes changed at version 3.16. Use `ViewKind()` to detect result and XY views if the script must be compatible with all versions of Spike2.

Here are some script definitions that may be useful, or at least save you some typing.

```
const kChanOff%=0, kWaveform%=1, kEventFall%=2, kEventRise%=3,
      kLevel%=4, kMarker%=5, kWaveMark%=6, kRealMark%=7,
      kTextMark%=8, kRealWave%=9, kXYChan%=120, kResultChan%= 127;

' This function convert a value returned by ChanKind() or SampleChanInfo()
' into a string representing the channel kind (useful for reports).
Func ChanKind$(kind%)
docase
case kind%=kChanOff% then return "Off";
case kind%=kWaveForm% then return "Waveform";
case kind%=kEventFall% then return "Event-";
case kind%=kEventRise% then return "Event+";
case kind%=kLevel% then return "Level";
case kind%=kMarker% then return "Marker";
case kind%=kWaveMark% then return "WaveMark";
case kind%=kRealMark% then return "RealMark";
case kind%=kTextMark% then return "TextMark";
case kind%=kRealWave% then return "RealWave";
case kind%=kXYChan% then return "XYChan";
case kind%=kResultChan% then return "Result";
else return "Unknown";
endcase;
end;
```

See also:

`ChanList()`, `MarkInfo()`, `MemChan()`, `SampleChanInfo()`, `ViewKind()`

ChanList()

This function fills an array with a channel list for a time, result or XY view. The result can be filtered to a subset of the available channels. There are two command variants:

Func ChanList(list%[{, types%});
Func ChanList(list%[], str\${, types%});

list% An integer array to fill with channel numbers. Element 0 is set to the number of channels returned. The remaining elements are channel numbers. If the array is too short, enough channels are returned to fill the array. It is unnecessary to list all the channel numbers for an XY view, since they are numbered contiguously.

types% This specifies the channels to return. If omitted, all channels are returned. Select channel types by adding the following codes, which are given as decimal and hexadecimal. XY views only support codes 1024 and 2048.

1 0x1	Waveform or result view channel
2 0x2	Event+ and Event- channels
4 0x4	Event +- channels (level data)
8 0x8	Marker channels

16 0x10	WaveMark data
32 0x20	TextMark data
64 0x40	RealMark data
128 0x80	Unused/deleted disk channels
256 0x100	Deleted channels on disk
512 0x200	Real wave channel

If none of the above values are used or this is an XY view, the list includes all channels. Add the following codes to exclude channels from the list:

1024 0x400	Exclude visible channels
2048 0x800	Exclude hidden channels
4096 0x1000	Exclude time view disk channels but not their duplicates
8192 0x2000	Exclude memory channels but not their duplicates
16384 0x4000	Exclude duplicated channels
32768 0x8000	Exclude selected channels
65536 0x10000	Exclude non-selected channels
2097152 0x200000	Exclude virtual channels but not their duplicates

`str%` A channel specifier such as "1..10,v1,m1" or "v1a". Only channels that exist in the current view are returned in `list%`. If `types%` is provided, only channels that match both the string and `types%` are returned in `list%`. If the specifier contains an error, channels up to the error are returned in `list%`, and the the function return value is set to 0.

Returns The number of channels that would be returned if the array were of unlimited length or 0 if the view is not the correct type or if a channel specifier was given that could not be parsed.

Number of channels in a data file

You can use this command to find the number of channels in a data file:

```
var list%[1], nCh%;
nCh% := ChanList(list%, 0x2063ff); 'number of channels in a data file
```

This works by counting channels of all types and unused and deleted channels, but excludes memory channels, duplicated channels and virtual channels.

See also:

`ChanKind()`, `ChanOrder[]`, `ChanShow()`, `DlgChan()`

ChanMeasure()

This takes a wide range of measurements on a channel in a time or result view. It is equivalent to the interactive cursor regions measurements, but takes the time range to process from user-supplied arguments rather than cursor positions. If the function you want is not provided here you will need to use `ChanData()` to get the individual data values.

```
Func ChanMeasure(chan%, type%, sPos, ePos{, &data%{, kind%}});
```

`chan%` The number of the channel to measure.

`type%` The type of measurement to take, see the documentation of the **Cursor Regions** window for details of these measurements. Note that codes 1 and 5 are the same when used from a script as there is no area to subtract as there can be in the **Cursor Regions** dialog. The possible values are:

1 Area	5 Area (scaled)	9 Minimum	13 Abs max.	17 Mean in X
2 Mean	6 Curve area	10 Peak to Peak	14 Peak	18 SD in X
3 Slope	7 Modulus	11 RMS Amplitude	15 Trough	19 Mean of abs
4 Sum	8 Maximum	12 Standard deviation	16 RMS error	

`sPos` The start position for the measurement in x axis units. If `sPos` is greater than or equal to `ePos`, the result is 0. In a result view, it is a common mistake to use `Cursor(1)` when you meant `BinToX(Cursor(1))` as the `Cursor()` command in a result view returns a bin number, not an x

axis position.

- ePos The end position in x axis units. In a result view, this is converted to a bin number which must be greater than the bin number obtained from sPos.
- data% Optional variable returned as 1 if a valid result was obtained or as 0 if there is no result (equivalent to a blank cell in the Cursor Regions dialog).
- kind% This optional variable forces a WaveMark channel to be treated as a waveform or as events for this measurement. If 0 or omitted, the channel is treated as a waveform if drawn in waveform, WaveMark or Cubic Spline mode, otherwise it is treated as an event channel. 1 forces waveform and 2 forces event.

Returns The function returns the requested measurement value.

See also:

ArrSum(), ChanData(), ChanValue(), Count(), Cursor(), XToBin()
 Cursor Regions dialog, Waveform channels and Result view measures, Event measures

ChanNew()

This function creates a new channel in the current time view. Unlike a memory channel, the created channel is permanent; any data written to it occupies disk space. You can use this to create channels for use with ChanWriteWave() and ChanSave().

```
Func ChanNew(chan%, type%{, size%{, binsz{, pre%{, trace%}}});
```

chan% The channel number to use in the range 1 to the number of channels in the data file (usually 32) or 0 for the first unused disk-based channel. You cannot use this routine to overwrite an existing channel; use ChanDelete() to remove it first.

type% The type of channel to create. Codes are:

- | | | | | | |
|---|----------------|---|-----------------|---|-----------|
| 1 | Waveform | 4 | Level (Event+-) | 7 | RealMark |
| 2 | Event (Event-) | 5 | Marker | 8 | TextMark |
| 3 | Event (Event+) | 6 | WaveMark | 9 | Real wave |

size% Used for TextMark, RealMark and WaveMark channels to set the maximum number of characters, reals or waveform points to attach to each item. Historically, with 32-bit data files, this used to set the disk buffer size in bytes for other channel types; we recommend that you use 0 for the default buffer size. You can omit this for channel types 2, 3 and 4.

binsz Used for waveform and WaveMark data to specify the time interval between the waveform points. This is rounded to the nearest multiple of the underlying time resolution. If you set this 0 or negative, the smallest bin size possible is set. You can omit this for types 2, 3 and 4.

If binsz is not a multiple of the underlying time resolution times *time per ADC* for the file, versions of Spike2 before 4.03 cannot read it.

pre% This must be present for WaveMark data to set the number of pre-trigger points.

trace% Optional, default 1, the number of interleaved traces for WaveMark data.

Returns The channel number if the channel was created, or a negative error code.

The channel has default title, units and comment. Use ChanTitle\$(), ChanUnits\$(), ChanComment\$(), ChanScale() and ChanOffset() to set these. The following code creates a copy of channel wFrom% (a waveform channel) in channel wTo%:

```
func CopyWave%(wFrom%, wTo%) 'Copy waveform to a memory channel
var err%, buffer%[8192], n%, sTime := 0.0;
if ChanKind(wFrom%)<>1 then return -1 endif; 'Not a waveform!
if ChanKind(wTo%)<>0 then ChanDelete(wTo%) endif; 'Check if used
ChanNew(wTo%, 1,0,BinSize(wFrom%)); 'Create waveform channel
if err%= wTo% then 'Created OK?
    ChanScale(wTo%, ChanScale(wFrom%)); 'Copy scale...
    ChanOffset(wTo%, ChanOffset(wFrom%)); '...and offset...
    ChanUnits$(wTo%, ChanUnits$(wFrom%)); '...and units
```

```

ChanTitle$(wTo%, "Copy");           'Set our own title
ChanComment$(wTo%, "Copied from channel "+Str$(wFrom%));
repeat
  n% := ChanData(wFrom%, buffer%[], sTime, MaxTime(), sTime);
  if n% > 0 then                     'read ok?
    n% := ChanWriteWave(wTo%, buffer%[:n%], sTime)
  endif;
  if n% > 0 then sTime += n% * BinSize(wTo%) endif;
until n% <= 0;
ChanShow(wTo%);                     'display new channel
endif;
return err%;   'Returns 0 if created OK
end;

```

See also:

ChanData(), ChanDelete(), ChanSave(), ChanWriteWave(), FileNew(), MemChan()

ChanNumbers()

This command shows and hides channel numbers in the current view and gets the channel number state. It is not an error to use this with data views that do not support channel numbers, but the command has no effect.

```
Func ChanNumbers( {show%} );
```

show% If present, 0 hides the channel number, and 1 shows it. Other values are reserved (and currently have the same effect as 1).

Returns The channel number display state at the time of the call.

See also:

YAxis(), YAxisMode()

ChanOffset()

Waveform and WaveMark data is stored as 16-bit integers with a scale factor and offset to convert to user units; this function gets and/or sets the y axis value that corresponds to a 16-bit waveform data value of zero. RealWave channels (which store data as 32-bit floating point values) also have a scale and offset; in this case the scale and offset convert the data to 16-bit values when required. The y axis user units for a channel are:

$$y \text{ axis value} = (16\text{-bit value}) * \text{scale} / 6553.6 + \text{offset}$$

RealWave data is stored as 32-bit floating point values, but the channel scale and offset are still available and are used when the channel data needs to be converted into 16-bit integers:

$$16\text{-bit value} = (y \text{ axis value} - \text{offset}) * 6553.6 / \text{scale}$$

```
Func ChanOffset(chan% {,offset} );
```

chan% The channel number.

offset If present, this sets the new channel offset. There are no limits on the value.

Returns The channel offset at the time of the call for waveform, RealWave or WaveMark channels, or 0.

See also:

ChanCalibrate(), ChanScale(), Optimise()

ChanOrder()

This command changes the order of channels in a time or result view, groups channels with a y axis so that they share a common y axis, sets the channel sorting order and gets list of visible channels in screen order from top to bottom. There are four command variants.

Move channels

The first moves channels relative to a channel or to a channel position in the list of all channels in the view:

```
Func ChanOrder(dest%, pos%, cSpC);
```

dest% The destination channel number or a value less than 1 indicating the position in the list of all channels including hidden channels but excluding channels in **cSpC**. As a position, 0 is the top channel, -1 is the next, and so on. If there are **n** channels, values of **-n** or less are taken to mean the bottom channel.

pos% The drop position relative to the destination channel: -1=above, 0=on top, 1=below. If you drop between grouped channels, dropped channels become group members (as long as they have a y axis).

cSpC A channel specifier for the channels to move.

Returns The number of moved channels.

Get information and ungroup channels

This variant gets grouping information and can convert all the channels in a group to ungrouped channels.

```
Func ChanOrder(dest%, opt%);
```

dest% A channel number in the view that identifies a group. A group is either a single channel, or a set of channels that share a single y axis.

opt% 0=returns the number of channels in the group that **dest%** belongs to or 0 if not grouped. 1-**n** returns the channel number of the **n**th channel in the group or 0 if no channel. -1 ungroups the group and returns the number of changed channels.

Returns The number of channels in the group, the **n**th channel in the group, or the number of changed channels, depending on **opt%**.

Sort channels

The third command variant sorts channels into numerical order.

```
Func ChanOrder(order%);
```

order% Set -1 for low, 1 for high numbered channels at the top and 0 to use the default channel ordering set by the Edit menu Preferences.

Returns -1 if low-numbered channels or 1 if high-numbered channels were previously at the top.

Get ordered channel lists

The fourth variant gets channel lists, ordered from top to bottom of all channels, all channels that start a group, and all channels within the **n**th group.

```
Func ChanOrder(list%[{, sel%});
```

list% An array that is filled with a channel list. **list%**[0] holds the number of items that follow in the list. If **list%**[0] holds **n**, **list%**[1] to **list%**[**n**] hold channel numbers. The returned channel numbers are in display order, with channels at the top of the screen first.

sel% Optional, taken as 0 if omitted. If 0, the list contains all the visible channels in the view. If -1, the list contains the first channels of each group, so if there are no groups, **list%**[0] holds 0. If **sel%** is greater than 0, say **n**, the list is returned holding all the channels in the **n**th group.

Returns The number of channels that match the option. This can be greater than **list%**[0] if the **list%** array is too small.

See also:

Channel specifiers, ChanList(), ChanWeight(), ViewStandard(), Yaxis(), YaxisLock()

ChanPenWidth()

This sets and gets the pen width for a channel in a Time or Result view; see the `XYDrawMode()` command for XY views. This script command is the equivalent of the View menu channel Pen Width dialog.

```
Func ChanPenWidth(cSpc{, new});
```

`cSpc` A channel specifier for one or more channels or -1 for all, -2 for visible and -3 for selected channels.

`new` If present, the new pen width for the specified channels, in points. A point is 1/72 of an inch, which is at least 1 screen pixel. If you set a negative width, the channel uses the pen width set in the Edit menu Preferences for data. A width of zero sets the thinnest pen (1 pixel) possible.

Returns The command returns the current pen width setting for the first channel that exists in the channel specification, or 0 if no channel exists.

See also:

Channel specifiers, Edit menu preferences, Pen width dialog, `XYDrawMode()`

ChanPixel()

This command reports the number of x and y user units that correspond to one pixel on the screen. Normally the x and y axes are in linear mode, but they can be set to logarithmic mode, in which case the returned values are the log increment that corresponds to one pixel on the screen.

```
Func ChanPixel(chan%, &x, &y);
```

`chan%` The channel in the current Time, Result or XY view.

`x` This real variable is set to the number of user units that are equivalent to a movement of one pixel to the right. This will correspond to the view x axis except for a WaveMark channel drawn in Overdraw WM mode, where the result is the time offset from the start of the WaveMark.

`y` This real variable is set to the number of user units that are equivalent to a movement of one pixel up. If there is no y axis, this is set to 0.

Returns A set of flags indicating which values were returned and if the units were modified. Flag values are:

- 1 The x axis value is set (this should always be set)
- 2 The y axis value is set (not set if the channel does not have a y axis)
- 4 The x axis is in log mode, so the x value is a log increment per pixel
- 8 The y axis is in log mode, so the y value is a log increment per pixel

You will need this information if you use the `ToolBarMouse()` or `DlgMouse()` function to get the mouse position and want to locate data elements that are close to the mouse pointer.

See also:

`DlgMouse()`, `ToolBarMouse()`

ChanPort()

This returns the physical hardware port that sampled data in a time view channel.

```
Func ChanPort(chan%);
```

`chan%` The channel number in the current time view.

Returns The physical data port or -1 if the view is not a time view, or if the channel was not sampled. Both event and ADC ports are enumerated from 0. WaveMark channels with multiple traces will report the port of the first trace.

See also:

`SampleEvent()`, `SampleWaveform()`, `SampleWaveMark()`

ChanProcessAdd()

This adds a channel process to a waveform or RealWave channel in a time view, matching the effect of the Add button in the Channel Process dialog.

```
Func ChanProcessAdd(chan%, PType% {, arg1, arg2, ...});
```

chan% A waveform or RealWave channel number in the current time view.

PType% The process type. The following process types are currently defined:

- 0 Rectify. Positive values are unchanged, negative values are negated. Data from waveform channels with a non-zero channel offset may be limited.
- 1 Smooth. This has one argument, a time range in seconds. The output at time t is the average of the input data from times $t-arg1$ to $t+arg1$.
- 2 DC Remove. There is one argument, a time range in seconds. The output at time t is the input minus the mean input from $t-arg1$ to $t+arg1$.
- 3 Slope. There is one argument, a time range in seconds. The slope at time t is calculated from the points in the time range $t-arg1$ to $t+arg1$. If you apply this process the channel scale and y axis units change and the channel offset becomes 0. The output is in input units per second.
- 4 Time shift. There is one argument, a time shift in seconds. A positive value shifts the trace right in the window, a negative value shifts it left.
- 5 Down sample. There is one argument, the down sample ratio.
- 6 Interpolate. Argument 1 is the sample interval, 2 is the alignment.
- 7 Chan match. Argument 1 is the channel to match.
- 8 RMS amplitude. Argument 1 is the time range.
- 9 Median filter. Argument 1 is the time range.
- 10 Fill gaps. Argument 1 is the maximum gap to interpolate across, 2 is the fill level for larger gaps.
- 11 Skip NaN. Removes non-numbers from RealWave data, leaving gaps.

arg# Optional process arguments to match the Channel Process dialog arguments for each process type. Each process has default arguments. It is an error to provide too many arguments or for an argument to be incorrect for the process.

Returns The index of the added process in the list of processes for the channel.

See also:

More about channel processes, ChanProcessArg(), ChanProcessClear(), ChanProcessInfo()

ChanProcessArg()

This gives you access to the parameters of channel processes added by the Channel Process dialog or by the ChanProcessAdd() command.

```
Func ChanProcessArg(chan%, id% {,n% {,arg}});
```

chan% The channel number in the current time view.

id% The process index. The first one added is number 1, the second is 2, and so on.

n% An optional argument number. The first argument is 1, the second 2, and so on.

arg If present, changes process argument n%. Ignored if out of range or illegal.

Returns If n% is omitted the function returns the number of arguments process id% uses. If n% is present it returns the value of argument n% at the time of the call.

See also:

ChanProcessAdd(), ChanProcessClear(), ChanProcessInfo()

ChanProcessClear()

This removes one or all processes from a channel or all processes from all channels.

```
Func ChanProcessClear({chan% {, id%}});
```

chan% A channel number in the current time view. If you omit this argument, or set it to -1, all processes for all channels are removed.

id% The index of the process to delete. Set -1 to delete all processes for channel **chan%**. After deleting, any remaining processes are renumbered.

Returns 0 if at least one process was deleted, -1 if no process was deleted.

See also:

ChanProcessAdd(), ChanProcessArg(), ChanProcessInfo()

ChanProcessCopy()

This copies all the channel processing from a source channel to a list of target channels, or clears all processing from a list of channels. You might use this when setting up a group of similar channels by setting the channel processes for the first channel, then copying them to the other channels.

```
Func ChanProcessCopy({cSpc%{, src%}});
```

cSpc A channel specifier for the target channels or 1 for all, -2 for visible and -3 for selected channels.

src% If omitted or 0, all processing for the channels defined by **cSpc** is cleared. Otherwise it is the channel to copy the processing from.

Returns 0 if at least one process was deleted, -1 if no process was deleted.

See also:

Channel specifiers, ChanProcessAdd(), ChanProcessArg(), ChanProcessClear(), ChanProcessInfo()

ChanProcessInfo()

This returns information about processes attached to a channel.

```
Func ChanProcessInfo(chan% {,id% {, arg%}});
```

chan% The channel number in the current time view.

id% Omit this to return the count of processes for the channel or set it to the index of the process to return information on. The first added process is number 1.

arg% Omit this to return the type of process index **id%**; see ChanProcessAdd() for the type codes. If present, the command returns the type of argument **arg%**: 0=integer, 1=real, 2=string (unused), 4=channel. The first argument is number 1.

Returns The information requested or a negative error code.

See also:

ChanProcessAdd(), ChanProcessArg(), ChanProcessClear()

ChanSave()

This function copies data with an optional time shift from source channels in the current time view to disk-based or memory buffer destination channels in any time view. If a destination channel exists, ChanSave() adds data to it, otherwise it creates a disk-based channel to match the source channel. You can set the number of disk-based channels in a new data file with the **nChans%** argument of FileNew().

The source data must be compatible with the destination. All event-based data types are mutually compatible. Destination data that is not present in the source is set to 0. Event times are set as close to the original times as the time bases of the source and destination views allow. If the conversion causes multiple events to fall at the same time, the converter adds 1 tick to separate the times. However, it will not do this for more consecutive events that the ratio of the two time bases.

If the source channel has a marker filter set, only events in the source channel that match the marker filter are copied to the destination channel. If the destination channel has a marker filter, the filter is unchanged for an existing channel and is set to accept all data for a new channel.

Adc and RealWave channels are compatible; conversion between them uses the channel scale and offset values. If the source and destination sample rates do not match, the data is interpolated using cubic splines. This is also done for non-matching WaveMark data.

If a destination disk-based channel already contains data, any added event-based data must be written after all data already contained in the channel. You can over-write existing waveform data with the same restrictions as for ChanWriteWave().

```
Func ChanSave(cSpc, dest%{, dh%{, sTime{, sTo{, dTime}}});
```

cSpc A channel specifier for the source channels or -1 for all, -2 for visible and -3 for selected channels. If you set multiple channels, **dest%** must be set to 0 or -1.

dest% This sets the destination channel or channels in the view identified by **dh%**. If **cSpc** refers to a single channel, this can be a channel number of a disk-based channel or an existing memory channel; otherwise **dest%** must be 0 or -1.

If **dest%** is 0, the lowest unused disk-based channels are used. If **dest%** is -1, the same channel number as the source is used. If the destination channel does not exist, it is created using the source channel settings.

dh% The handle of the destination view. Omit or set 0 to use the current time view.

sTime The start time to read source data. This is 0.0 if omitted.

sTo Source data is read up to this time. If omitted, the end of the data is used.

dTime The destination time. If this is omitted, **sTime** is used. The source data is time shifted by **dTime-sTime** before being copied to the destination channels.

Returns With a single source channel, the result is the destination channel number. With multiple source channels, the result is the number of channels copied without error. Negative results are: -1 = **cSpc** illegal, -2 = **dest%** illegal, -3 = **dh%** is not a time view handle, -4 = destination channel could not be created or was not compatible with the source, -5 = a file system error occurred when copying data.

Append files example

This command has many applications. This example creates a new file and adds the contents of two data files to it. We assume the files hold the same types of channels. In this case we create an output file with the same time resolution as the first input file. This example does not check for errors (**fh%** should be > 0 after open, for example).

```
var fh%,dh%;      'file handles of the source and destination files
fh% := FileOpen("file1.smr", 0, 0);      'open file1, assumes fh% OK
dh% := FileNew(7, 1, View(fh%).Binsize()*1e6, 1, 1); 'make output
View(fh%);ChanSave(-1, -1, dh%);FileClose(0,-1);  'copy and close
fh% := FileOpen("file2.smr", 0, 0);      'open file2 and append
ChanSave(-1, -1, dh%, 0, MaxTime(), View(dh%).MaxTime()+1.0);
FileClose(0,-1);      'close file2
View(dh%);ChanShow(-1);Draw(0, MaxTime());      'Display the result
```

Resample Waveform channels

If you want to replay an arbitrary waveform through the 1401 DACs while sampling, the waveform must have a sample interval that the 1401 can achieve. For example, suppose that you want to replay a sound waveform that was sampled at 44.1 kHz. This is a sample interval of approximately 22.676 microseconds, which is not achievable with the 1401 DACs. When you import this data, the underlying time base of the file will be set to a non-integral number of microseconds so that the data will display correctly. One way around this is to resample the data to a 1 microsecond time base:

```
var fh%,dh%;      'file handles of the source and destination files
fh% := FileOpen("file1.smr", 0, 0); 'open file1 with 44.1 kHz data
dh% := FileNew(7, 1, 1, 1, 1);      'make output, 1 us clock ticks
View(fh%);ChanSave(-1, -1, dh%);FileClose(0,-1); 'copy and close
View(dh%);ChanShow(-1);Draw(0, MaxTime());      'Display the result
```

Beware files with more than 32 channels

If the source channel has more than 32 channels, you will need to modify the FileNew() commands in the previous examples:

```
var fh%,dh%;      'file handles of the source and destination files
fh% := FileOpen("file1.smr", 0, 0);      'open file1
var chans%[1], nCh%; 'dummy list, number of channels in the file
nCh% := ChanList(list%, 0x2063ff); 'get number of channels
if (nCh% < 32) then nCh% := 32 endif;      'Make sure valid
dh% := FileNew(7, 1, 1, 1, 1, nCh%);      'create file, nCh% channels
...
```

Big files and 64-bit files

In addition to the extra argument to FileNew() to set the number of channels, there is the big% argument to control the file type. Setting the file type determines how big the file can be in both time and disk space. If you set a 64-bit file, only Spike2 version 8 can read the result. If you set a 32-bit "big file", Spike2 version 6 onwards can read it but only 7 onward can modify it.

See also:

Channel specifiers, ChanNew(), ChanOffset(), ChanScale(), ChanWriteWave(), FileNew()

ChanScale()

In a time view, this function gets and/or sets the scaling between the 16-bit integer data used to store waveform and WaveMark data and the real units of the channel. RealWave channels (which store data as 32-bit floating point values) also have a scale and offset; in this case the scale and offset convert the data to 16-bit values when required. The 16-bit waveform data has values between +32767 and -32768. The y axis user units for a channel are:

$$y \text{ axis value} = (16\text{-bit value}) * \text{scale} / 6553.6 + \text{offset}$$

With the standard ± 5 Volt 1401 ADC inputs, 6553.6 is equivalent to 1 Volt at the 1401 input. In this case, a scale of 1.0 and an offset of 0.0 gives a y axis calibrated in Volts.

```
Func ChanScale(chan% {,scale});
```

chan% The channel number.

scale If present, sets the channel scaling. We suggest you don't set 0.0 as a scale!

Returns The scale at the time of the call for waveform or WaveMark channels, or 0.

You can set the channel scaling interactively from the Channel Information dialog.

See also:

ChanCalibrate(), ChanOffset(), Optimise()

ChanSearch()

This searches a time or result view channel for a user-defined feature. It is the same as an active cursor search, but does not use or move cursors. Searches are done in the current channel drawing mode. WaveMark data drawn as a waveform is searched as a waveform; to search it as events, set an event drawing mode. There is no need for the channel to be visible; you only need to set the display mode before searching.

```
Func ChanSearch(chan%, mode%, sT, eT{, sp1{, sp2{, width{, flag%{, lv2}}}});
```

chan% The number of a channel in the time or result view to search.

- mode%** This sets the search mode, as for the active cursors. Modes in italics (15 and 16) cannot be used: mode 16 is not relevant and mode 15 can be emulated by mode 6, 7 or 8. See the cursor mode dialog documentation for details of each mode.
- | | | |
|--------------------|-------------------------|----------------------------|
| 1 Maximum value | 8 Falling threshold | 15 <i>Repolarisation %</i> |
| 2 Minimum value | 9 Steepest rising | 16 <i>Expression</i> |
| 3 Max excursion | 10 Steepest falling | 17 Turning point |
| 4 Peak find | 11 Steepest slope (+/-) | 18 Slope% |
| 5 Trough find | 12 Slope peak | 19 Outside thresholds |
| 6 Threshold | 13 Slope trough | 20 Inside thresholds |
| 7 Rising threshold | 14 Data points | |
- sT, eT** The search start and end positions (in seconds in a time view or bins in a result view). If eT is less than sT, the search is backwards.
- sp1** This is the amplitude for peaks, threshold level for threshold crossings, baseline level for maximum excursion and number of points for mode 14. It is in the y axis units of the search channel (y axis units per second for slopes). If omitted, the value 0.0 is used. Set it to 0 if sp1 is not required for the mode.
- sp2** This is hysteresis for threshold crossings and percent for Slope%. If omitted, the value 0.0 is used. Set it to 0 if sp2 is not required for the mode.
- width** This is the width in seconds/bins for slope measurements. It sets the minimum time that the data must be above/below a level for threshold measures. It sets the maximum allowed width of a peak or trough. If omitted, 0 is used. Set 0 if width is not required for the mode or you do not want a time constraint.
- flag%** This is the sum of flag values and is 0 if omitted. At the moment, the only value defined is 1=ignore gaps in waveform data. If this value is not set, a search of a waveform channel will stop at a gap in the data.
- lv2** This is the second level for modes 19 and 20. If omitted, 0 is used.
- Returns** A positive time or bin number if the search succeeds, -1 if it fails or a negative error code.

See also:

Active cursors, Cursor mode dialog, ChanMeasure(), ChanValue(), CursorActive()

ChanSelect()

This function is used to report on the selected/unselected state of a channel in a time or result view, and to change the selected state of a channel.

```
Func ChanSelect(chan% {,new%});
```

- chan%** The channel number or -1 for all visible channels. Hidden channels and non-existent channels always appear to be unselected.
- new%** If present it sets the state: 0 for unselected, not 0 for selected. If omitted, the state is unchanged. Attempts to change invisible channels are ignored.

Returns The channel state at the time of the call, 0 for unselected, 1 for selected. If you set chan% to -1, the function returns the number of selected channels.

You can get a list of all selected channels with ChanList().

See also:

ChanHide(), ChanList(), ChanOrder(), ChanShow(), ChanWeight()

ChanShow()

Display a channel, or a list of channels in a time, result or XY view. Turning on a channel that is on has no effect. Turning on a channel that doesn't exist has no effect.

```
Proc ChanShow(cSpc {,cSpc...});
```

`cSpC` A channel specifier for the channels to show.

See also:

Channel specifiers, `ChanHide()`, `ChanList()`, `ChanVisible()`, View menu show/hide

ChanTitle\$()

This returns or sets the channel title string in a time, result or XY view. A title string is up to 9 characters long. In an XY view the channel titles are visible in the Key window.

Setting a duplicate channel title does not change the original channel. Setting the original channel title sets the titles of duplicates unless the duplicates have their own title. If you set the title of a duplicate to "", the title reverts to the title of the channel it duplicates.

```
Func ChanTitle$(chan%{,new$});
```

`chan%` The channel number. In XY views you can also use 0 to change the overall (displayed) y axis title.

`new$` An optional string holding the new channel title.

Returns The channel title string for the channel. If the channel does not exist, the function does nothing and returns an empty string.

See also:

`ChanComment()`, `XYKey()`

ChanUnits\$()

This gets or sets the units for waveform or WaveMark channels and result or XY views. From 5.03 you can read back and set the displayed units of time view event-based channels. Changes to event-based channels are lost when the display mode changes.

```
Func ChanUnits$(chan%{,new$});
```

`chan%` A channel in the time or result view. In an XY view the channel number is ignored. We suggest you use 0 to match `ChanTitle$()`.

`new$` An optional string holding the new y axis units for the channel (up to 5 characters). If the string is too long, it is truncated.

Returns It returns the old units of the designated channel. If the channel does not exist or is not of a suitable type, the function does nothing and returns an empty string.

See also:

`ChanScale()`, `ChanOffset()`, `ChanTitle$()`

ChanValue()

In time views this returns the value of a channel at a time. For a waveform channel it returns the waveform value, for other channel types, it returns a value in the y axis units of the channel display mode. If the channel has no y axis or is drawn in raster mode, the value is the time of the next event on the channel.

In a result view, this returns the value of the result corresponding to an x axis value. You can use the `[bin]` notation to access result views by bin number.

```
Func ChanValue(chan%,pos{,&data%{,mode%{,binsz{,trig%|edge%{,as%}}}}});
```

`chan%` The channel number in the time or result view.

`pos` In a time view, the time for which the value is needed. In a result view, this is the x axis value for which a result is needed.

`data%` Returned as 1 if there is data at `pos`, 0 if not. For example, for a waveform if there was no data within `Binsize(chan%)` of `pos`, this would be set to 0.

- mode%** If present, this sets the display mode in which to read the value from a time view. If **mode%** is inappropriate or absent, the current display mode is used. This parameter is ignored in a result view. See `DrawMode()` for a full description of all drawing modes. The modes in a time view are:
- 0 The current mode for the channel. Any additional arguments are ignored.
 - 1 Dots mode for events. Returns the event time at or after **pos**.
 - 2 Lines mode, result is the same as mode 1.
 - 3 Waveform mode.
 - 4 WaveMark mode, returns waveform values for WaveMark channels.
 - 5 Rate mode. The **binSz** argument sets the width of each bin.
 - 6,11 Mean frequency mode, **binSz** sets the time period, 11 is rate per minute.
 - 7,12 Instantaneous frequency, returns value at next event, 12 is per minute.
 - 8 Raster mode, **trig%** sets the trigger channel, result as for mode 1.
 - 13 Cubic spline for waveforms and WaveMark data.
 - 16 Skyline mode for waveform, RealWave and RealMark channels.
- binSz** This sets the width of the rate histogram bins and the smoothing period for mean frequency mode when specifying your own mode.
- trig%** The trigger channel for raster displays, we assume raster displays of level data are never required.
- edge%** For level data event channels. This sets which edges of the signal to use for mean frequency, instantaneous frequency and rate modes: 0=both edges, 1=rising edges, 2=falling edges. If **edge%** is omitted, both edges are used.
- as%** Used with instantaneous frequency mode to determine how the data is measured. 0=Default, 1=Dots, 2=Line, 3=Skyline.
- Returns** It returns the value or 0 if no data is found. For waveform data, if there is no data within `Binsize(chan%)` of the time, the value is zero.
- If **data%** is omitted any error stops the script. Errors include: no current window, current window not a time or result view, no data at **pos**, and **pos** beyond range of x axis. If **data%** is present, errors cause it to be set to 0.

See also:`DrawMode()`, `ChanData()`, `ChanMeasure()`, `Cursor Values`, `MinMax()`

ChanVisible()

This returns the state of a channel in a time, result or XY view as 1 if the channel is visible and 0 if it is not. If you use a silly channel number, the result is 0 (not displayed).

```
Func ChanVisible(chan%);
```

chan% The channel to report on.

Returns 1 if the channel is displayed, 0 if it is not.

See also:`ChanShow()`, `ChanHide()`

ChanWeight()

This function sets the relative vertical space to give a channel or a list of channels. The standard vertical space corresponds to a weight of 1. When Spike2 allocates vertical space, channels are of two types: channels with a y axis and channels without a y axis. Spike2 calculates how much space to give each channel type assuming all channels have a weight of 1. Then the actual space allocated is proportional to the standard space multiplied by the weight factor. This means that if you increase the weight of one channel, all other channels get less space in proportion to their original space.

```
Func ChanWeight(cSpc{, new});
```

cSpc The channel specifier for the list of channels to process.

new If present, a value between 0.001 and 1000.0 that sets the weight for all the channels in the list. Values outside this range are limited to the range.

Returns The command returns the channel weight of the first channel in the list.

See also:

Channel specifiers, `ChanOrder()`, `ViewStandard()`

ChanWriteWave()

This function writes real or integer data to a waveform (16-bit integer) or RealWave (32-bit floating point) channel. The time gap between array points is the `Binsize()` value of the channel. You can overwrite existing data and add data to the end of the channel. You cannot fill in gaps in wave channels; values written into gaps in previously written data are ignored.

```
Func ChanWriteWave(chan%, arr[]|arr%[], sTime);
```

chan% The waveform or RealWave channel in the current time view to write data to. This can be a duplicate channel, a disk channel or a memory channel. If you write to a duplicated channel, the original channel data is changed.

arr A real or integer array to write to the channel. When writing real data to a waveform channel or integer data to a RealWave channel, the data is converted to match the channel format using the channel scale and offset. When writing to a waveform, output is limited to 16-bit integers in the range -32768 to 32767.

sTime The first array point time. When overwriting, if the time does not align with existing data it is reduced by less than one sample interval to align it.

Returns The number of points processed including points skipped due to gaps in existing channel data or a negative error code, for example if the file is read-only.

The function will cause a fatal script error if used on the wrong view type, the wrong channel type or if the system runs out of memory.

See also:

`Binsize()`, `ChanData()`, `ChanNew()`, `ChanOffset()`, `ChanScale()`

Chr\$()

This function converts a code to a character and returns it as a single character string.

```
Func Chr$(code%);
```

code% The code to convert. Codes that have no character representation will produce unpredictable results.

Returns A string holding one character to represent the code.

See also:

`%c` format in `Print()`, `Asc()`, `DelStr$()`, `LCase$()`, `Left$()`, `Len()`, `Mid$()`, `Print$()`, `Right$()`, `Str$()`, `UCase$()`, `Val()`

Colour()

Deprecated. This is provided for backwards compatibility with old scripts. Use `ColourGet()` and `ColourSet()` in new scripts.

This function gets and/or sets the colours of items using the colour palette. XY channels are coloured using `XYColour()`. The colours set for time and result view drawing modes can be overridden by `ChanColour()`.

```
Func Colour(item% {,col%});
```

item% This selects the item to be coloured. The available items are Given in the table. Note that this table

matches the `ColourSet()` table only as far as item 17.

0 Channel number	14 Result dots	28 Cursors
1 Time background	15 Result skyline	29 Controls (not used)
2 Waveform channel	16 Result histogram	30 Grid colour
3 Events as dots	17 Result histogram fill	31 X and Y Axes
4 Events as lines	18 Marker code 00	32 XY background
5 Level events	19 Marker code 01	33 Not saving to disk
6 Marker data	20 Marker code 02	34 Result SEM and SD
7 Mean frequency	21 Marker code 03	35 Fitted curves
8 Inst. Frequency	22 Marker code 04	36 Cluster background
9 Raster dots	23 Marker code 05	37 WaveMark background
10 Rate outline	24 Marker code 06	38 Vertical markers
11 Rate fill	25 Marker code 07	39 XY channels
12 Result background	26 Marker code 08	40 XY fill
13 Result lines	27 TextMark text	41 XY key labels

`col%` If present, this sets the index of the colour in the colour palette to be applied to the item. There are 40 colours in the palette, numbered 0 to 39.

Returns The index into the colour palette of the colour that is nearest to the the colour of the item at the time of the call. **Beware:** if you set a colour index for an item and read it back you may not get the index you set if another index holds the same colour.

See also:

Colour dialog, `ChanColour()`, `PaletteGet()`, `PaletteSet()`, `ViewColour()`, `ViewUseColour()`, `XYColour()`

ColourGet()

Get the RGB colour for an item in the palette, the main colour table or the marker colour table. Added at version 7.07.

```
Func ColourGet(table%{, item%, &r, &g, &b});
```

`table%` -1 to select the colour palette, 0 for the main colour table, 1 for the marker table.

`item%` The item number in the table selected by `table%`. See `ColourSet()` for a description of the item numbers.

`r g b` If present, returned as the red, green and blue colour values in the range 0.0 to 1.0.

Returns If only the table number is supplied, returns the length of the table, otherwise 0.

See also:

Colour dialog, `ChanColourGet()`, `ColourSet()`, `ViewColourGet()`

ColourSet()

Set the RGB colour for an item in the palette, the main colour table or the marker colour table. If you change the colour of any item that is visible, this will make the display invalid and the item will repaint at the next opportunity. This command was added at Spike2 version 7.07.

```
Func ColourSet(table%, size%);  
Func ColourSet(table%, item%, r, g, b);
```

`table%` -1 to select the colour palette, 0 for the main colour table, 1 for the marker table.

`size%` Used to set the size of the marker colour table (`table% = 1`) in the range 3 to 256. The initial, default value is 9 for marker colours 0-8. If you increase the size, the new table elements are set to black. Setting a size of 0 resets the table to the default size. Setting a size of -1 resets the table size and sets default colours. You can use -1 with any table; setting the table size only works for the marker colour table.

`item%` The item number in the table selected by `table%`. This is an index from 0 to 39 for the colour palette. It is one of the following for the main colour table:

0 Channel numbers	11 Rate fill	22 X and Y Axes
1 Time background	12 Result background	23 XY background
2 Waveform channel	13 Result lines	24 Not saving to disk
3 Events as dots	14 Result dots	25 Result SEM and SD
4 Events as lines	15 Result skyline	26 Fitted curves
5 Level events	16 Result histogram	27 Cluster background
6 Marker data and code	17 Result histogram fill	28 WaveMark background
7 Mean frequency	18 TextMark text	29 Vertical markers
8 Inst. Frequency	19 Cursors	30 XY channels
9 Raster dots	20 Controls (unused)	31 XY fill
10 Rate outline	21 Grid colour	32 XY key labels

For the marker colour table it is an index from 0 up to the table size -1.

`r g b` If present, sets the red, green and blue values in the range 0 to 1.0.

Returns If only the table number is supplied, returns the length of the table, or 0.

See also:

Colour dialog, `ChanColourSet()`, `ColourGet()`, `ViewColourSet()`, `ViewUseColour()`

Conditioner commands

The `Cond...` family of commands control signal conditioners. They support the CED 1902 programmable signal conditioner, the Power 1401 programmable gain options and the Axon Instruments CyberAmp. Other conditioners may be added in the future.

These commands do not select which serial port (if any) the conditioner uses or the type of conditioner supported. You choose the conditioner type when you install Spike2. You set the serial port in the Edit menu Preferences option. All these commands require a `port%` argument. This is the physical waveform input port number that the conditioner is attached to. It is not a channel number in a time view.

You can access the built-in interactive support for the conditioner from the Sampling Configuration channel parameters dialog. This can be a useful short-cut to getting the lists of gains and signal sources available on your conditioner(s).

See also:

`CondFeature()`, `CondFilter()`, `CondFilterList()`, `CondFilterType()`, `CondGain()`, `CondGainList()`, `CondGet()`, `CondOffset()`, `CondOffsetLimit()`, `CondRevision$()`, `CondSet()`, `CondSourceList()`, `CondType()`

CondFeature()

This command gets and sets special signal conditioner features that are not general enough to have dedicated commands to support them. See the `CondSet()` command for more details of conditioner operation. There are four command variants:

Get feature count

```
Func CondFeature(port%);
```

`port%` The waveform port number that the conditioner is connected to.

Returns The number of special features supported by the signal conditioner.

Get feature information

The command supports two types of features: those that have a set of discrete values such as ["None", "Rectify"], and those that support a continuous range of floating point values, such as 10.0 to 25.6, for

example. To determine if the feature is continuous or discrete, call the function with only the first 3 or 4 arguments.

```
Func CondFeature(port%, feat%, &name${,&flags%{, list${[]}}});  
Func CondFeature(port%, feat%, &name${,&flags%{, &low{, &high}}});
```

feat% The feature number, from 1 to the number of features available

name\$ Returned set to the name of the feature.

flags% Returned set to the feature flags. Currently, none are defined, so this will be 0.

list\$ Returned set to an array of the possible settings (strings) for discrete type.

low Returned as the low limit for a feature with continuous values.

high Returned as the upper limit for a feature with continuous values

Returns The number of discrete feature values, or 0 if the feature supports continuous values over the range returned in **low** and **high**.

Set feature value

```
Func CondFeature(port%, feat%{, val});
```

val If present, it sets the value for the feature set by **feat%** and **port%**. If this is a continuous feature, **val** sets the new value. If the feature has **n** discrete setting, **val** should be 0 to **n-1** to select the feature corresponding to the feature description in **list\$[val]**. If **val** exceeds the allowed range, a continuous feature is set to the nearest allowed value and a discrete feature is unchanged.

Returns The feature value at the time of the call (before any change). This is an integer index for features with discrete values otherwise it is the feature value.

See also:

`CondFilterList()`, `CondGain()`, `CondGainList()`, `CondGet()`, `CondOffset()`,
`CondOffsetLimit()`, `CondRevision$()`, `CondSet()`, `CondSourceList()`, `CondType()`

CondFilter()

This sets or gets the frequency of the low-pass or high-pass filter of the signal conditioner. See the `CondSet()` command for more details of conditioner operation.

```
Func CondFilter(port%, high% {,freq{, type%}});
```

port% The waveform port number that the conditioner is connected to.

high% This selects which filter to set or get: 0 for low-pass, 1 for high-pass.

freq If present, this sets the desired corner frequency of the selected filter. See the `CondSet()` description for more information. Set 0 for no filtering. If omitted, the frequency is not changed. The high-pass frequency must be set lower than the frequency of the low-pass filter, if not the function returns a negative code.

type% Optional, taken as 1 if omitted. The filter type to use when setting the filter.

Returns The cut-off frequency of the selected filter at the time of call, or a negative error code. A return value of 0 means that there is no filtering of the type selected.

See also:

`CondFilterList()`, `CondGain()`, `CondGainList()`, `CondGet()`, `CondOffset()`,
`CondOffsetLimit()`, `CondRevision$()`, `CondSet()`, `CondSourceList()`, `CondType()`

CondFilterList()

This function gets a list of the possible filter frequencies of the conditioner. Conditioners that support continuous frequency ranges also supply a list of frequencies to match the list of frequencies shown in the conditioner control panel. See the `CondSet()` command for more details of conditioner operation.

```
Func CondFilterList(port%, high%, freq[]{, type%});
```

port% The waveform port number that the conditioner is connected to.

high% Selects which filter to get: 0 for low-pass, 1 for high-pass.

freq[] An array of reals holding the cut-off frequencies of the selected filter. There is always a value of 0 meaning no filtering.

type% Optional, taken as 1 if omitted. The filter type in the range 1 to the number of types (as returned by CondFilterType()).

Returns The number of filter frequencies (including 0) or a negative error code.

See also:

CondFilter(), CondFilterType(), CondGainList(), CondGet(), CondOffset(), CondOffsetLimit(), CondRevision\$(), CondSet(), CondSourceList(), CondType()

CondFilterType()

This function returns information about the filter types supported by the conditioner for the low pass and high pass filters. For example, the CED 1902 mk IV supports a choice of filter types. There are three command variants:

Get number of filter types

```
Func CondFilterType(port%, high%);
```

port% The waveform port number that the conditioner is connected to.

high% Selects which filter to return information for: 0 for low-pass, 1 for high-pass.

Returns The number of filter types.

Get filter type in use

```
Func CondFilterType(port%, high%, 0);
```

Returns The currently selected filter type, from 1 to the number of filter types.

Get filter type information

```
Func CondFilterType(port%, high%, type%{, &name${, &lower{, &upper}}});
```

type% The filter number, from 1 to the number of available filters.

name\$ If present, returned holding the name of the filter selected by type%.

lower If present, returned as the lowest filter frequency (excluding 0, meaning 'off').

upper If present, returned as the highest supported filter frequency.

Returns The number of frequency values the filter can be set to (including 0) or 0 if the filter corner frequency can be set to any value in the range lower to upper.

See also:

CondFilter(), CondGain(), CondGainList(), CondGet(), CondOffset(), CondOffsetLimit(), CondRevision\$(), CondSet(), CondSourceList(), CondType()

CondGain()

This sets and gets the gain of the signal passing through the signal conditioner See the CondSet() command for more details of conditioner operation.

```
Func CondGain(port% {, gain});
```

port% The waveform port number that the conditioner is connected to.

gain If present this sets the ratio of output signal to the input signal. If this argument is omitted, the current gain is returned. The conditioner will set the nearest gain it can to the requested value.

Returns The gain at the time of call, or a negative error code.

See also:

CondFilter(), CondFilterList(), CondGainList(), CondGet(), CondOffset(), CondOffsetLimit(), CondRevision\$, CondSet(), CondSourceList(), CondType()

CondGainList()

This function gets a list of the possible gains of the conditioner for the selected signal source. See the CondSet() command for more details of conditioner operation.

```
Func CondGainList(port%, gain[]);
```

port% The waveform port number that the conditioner is connected to.

gain[] An array of reals holding the conditioner gains for the selected signal source. If a conditioner (for example, 1902) has a fixed set of gains, this is the set of gain values. If the conditioner supports continuously variable gain, the first two elements of this array hold the minimum and the maximum values of the gain.

Returns The number of gain values if the conditioner has a fixed set of gains or 2 if the conditioner has continuously variable gain. In the case of an error, a negative error code is returned.

See also:

CondFilter(), CondFilterList(), CondGain(), CondGet(), CondOffset(), CondOffsetLimit(), CondRevision\$, CondSet(), CondSourceList(), CondType()

CondGet()

This function gets the input signal source of the signal conditioner, and the conditioner settings for gain, offset, filters and coupling. The settings are returned in arguments which must all be variables. See CondSet() for details of conditioner operation.

```
Func CondGet(port%, &in%, &gain, &offs, &low, &hi, &notch%, &ac%{, &typeL%{, &typeH%}});
```

port% The waveform port number that the conditioner is connected to.

in% Returned as the zero-based index of the input signal source (see CondSet()).

gain Returned as the ratio of output to input signal amplitude (ignoring filtering).

offs A value added to the input waveform to move it into a more useful range. Offset is specified in user units and is only meaningful when DC coupling is used.

low Returned as the cut-off frequency of the low-pass filter. A value of 0 means that there is no low-pass filtering enabled on this channel.

hi Returned as the cut-off frequency of the high-pass filter. A value of 0 means that there is no high-pass filtering enabled on this channel.

notch% Returned as 0 if the mains notch filter is off, and 1 if it is on.

ac% Returned as 1 for AC or 0 for DC coupling.

typeL% Optional integer variable returned holding the low-pass filter type number as described for CondFilterType().

typeH% Optional integer variable returned holding the high-pass filter type number.

Returns 0 if all well or a negative error code.

See also:

CondFilter(), CondFilterType(), CondFilterList(), CondGain(), CondGainList(),

CondOffset(), CondOffsetLimit(), CondRevision\$, CondSet(), CondSourceList(), CondType()

CondOffset()

This sets or gets the offset added to the input signal of the signal conditioner. See the CondSet() command for more details of conditioner operation.

Func CondOffset(port%, offs);

port% The waveform port number that the conditioner is connected to.

offs The value to add to the input waveform of the conditioner to move it into a more useful range. If this argument is omitted, the current offset is returned. The conditioner will set the nearest value it can to the requested value.

Returns The offset at the time of call, or a negative error code.

See also:

CondFilter(), CondFilterList(), CondGain(), CondGainList(), CondGet(), CondOffsetLimit(), CondRevision\$, CondSet(), CondSourceList(), CondType()

CondOffsetLimit()

This function gets the maximum and minimum values of the offset range of the conditioner for the currently selected signal source. See the CondSet() command for more details of conditioner operation.

Func CondOffsetLimit(port%, offs[]);

port% The waveform port number that the conditioner is connected to.

offs[] This is an array of real numbers returned holding the minimum (offs[0]) and the maximum (offs[1]) values of the offset range of the conditioner for the currently selected signal source.

Returns 2 or a negative error code.

See also:

CondFilter(), CondFilterList(), CondGain(), CondGainList(), CondGet(), CondOffset(), CondRevision\$, CondSet(), CondSourceList(), CondType()

CondRevision\$()

This function returns the name and version of the signal conditioner as a string or an empty string if there is no conditioner for the port.

Func CondRevision\$(port%);

port% The waveform port number that the conditioner is connected to.

Returns A string describing the conditioner. Strings defined so far include: "1902ssh", where ss is the 1902 ROM software version number and h is the hardware revision level; and "CYBERAMP 3n0 REV x.y.z" where n is 2 or 8.

See also:

CondFeature(), CondFilter(), CondFilterList(), CondFilterType(), CondGain(), CondGainList(), CondGet(), CondOffset(), CondOffsetLimit(), CondSet(), CondSourceList(), CondType()

CondSet()

This sets the input signal source, gain, offset, filters and coupling of the conditioner. All values are requests; the command sets the closest possible value to that requested. If it is important to know what has actually been set you should read back the values with `CondGet()` after setting them, or use the functions for reading specific values.

```
Func CondSet(port%, in%, gain, offs {,low, high, notch%, ac%{, typeL%{, typeH%}}});
```

port% The waveform port number that the conditioner is connected to.

in% A conditioner has one or more signal sources. For example, the CED 1902 supports `Grounded`, `Single ended`, `Normal Diff`, `Inverted Diff`, etc. Conditioners of the same type may have different sources. To select a source, set **in%** to its zero-based index in the list returned by `CondSourceList()`.

gain This is the desired ratio of output to the input signal amplitude (ignoring the effect of any filtering). The actual gain depends on the capabilities of the signal conditioner, see `CondGainList()`. The gain range may be altered by the choice of signal source. For example, the 1902 Isolated Amp input has a build-in gain of 100. This command sets the nearest gain to the requested value.

offs This is the desired value in user units to add to the input waveform to move it into a more useful range. Offsets are only meaningful with DC coupling. Different conditioners have different offset ranges, and the offset range may be altered by the choice of signal source, see `CondOffsetLimit()`. The command will set the nearest offset it can to the desired value.

low If present and greater than 0, it is the desired corner frequency of the low-pass filter. Low-pass filters are used to reduce the high frequency content of the signal, both to satisfy the sampling requirement, and in case where it is known that no useful information is to be found in the signal above a certain frequency. If omitted, or 0, there is no low-pass filtering. The actual filter value set depends on the capabilities of the signal conditioner.

high If present and greater than 0, it is the high-pass filter corner frequency. High-pass filters reduce the low-frequency content of the signal. This must be set lower than the frequency of the low-pass filter; if not, the function returns a negative code. If omitted, or set to 0, there is no high-pass filtering.

Different signal conditioners have different ranges of frequency filtering. To find out the real filter frequency set, use `CondFilter()`. `CondFilterList()` returns the list of possible filter frequencies.

notch% Some signal conditioners have a mains-frequency notch filter (usually 50 Hz or 60 Hz) used to reduce the effect of mains interference on low level signals. This filter will remove the fundamental 50 Hz or 60 Hz signal; it will not remove higher harmonics (for example 150 Hz). If **notch%** is present with a value greater than 0, the notch filter is on. If omitted, or 0, the notch filter is off.

ac% The 1902 supports both AC and DC signal coupling. If you set AC coupling you should probably set the offset to zero. If **ac%** is greater than 0, the signal conditioner is AC coupled. If omitted or 0, the signal conditioner is DC coupled.

typeL% Optional value, taken as 1 if omitted, that sets the low-pass filter type as described for `CondFilterType()` in the range 1 to the number of filter types.

typeH% Optional value, taken as 1 if omitted, that sets the high-pass filter type.

Returns 0 if all well or a negative error code.

See also:

`CondFilter()`, `CondFilterList()`, `CondFilterType()`, `CondGain()`, `CondGainList()`, `CondGet()`, `CondOffset()`, `CondOffsetLimit()`, `CondRevision$()`, `CondSourceList()`, `CondType()`

CondSourceList()

This function gets a list of the possible signal source names of the conditioner, or the specific signal source name with the given index number. See the `CondSet()` command for more details of conditioner operation.

```
Func CondSourceList(port%, src$[]|src$ {,in%});
```

`port%` The waveform port number that the conditioner is connected to.

`src$` This is either a string variable or an array of strings that is returned holding the name(s) of signal sources. Only one name is returned per string.

`in%` This argument lets you select an individual source or all sources. If present and greater than or equal to 0, it is the zero-based index number of the signal source to return. In this case, only one source is returned, even if `src$` is an array.

If omitted and `src$` is a string, the first source is returned in `src$`. If `src$[]` is an array of strings, as many sources as will fit in the string array are returned.

Returns If `in%` is greater than or equal to 0, it returns 1 or a negative error code. If `in%` is omitted, it returns the number of signal sources or a negative error code.

See also:

`CondFilter()`, `CondFilterList()`, `CondGain()`, `CondGainList()`, `CondGet()`, `CondOffset()`, `CondOffsetLimit()`, `CondRevision$()`, `CondSet()`, `CondType()`

CondType()

This function returns the type of the signal conditioner.

```
Func CondType(port%);
```

`port%` The waveform port number that the conditioner is connected to.

Returns A value indicating the connected conditioner or an error code:

- 4 Digitimer 360
- 3 Power1401 with gain controls
- 2 Axon Instruments CyberAmp
- 1 CED 1902
- 0 No conditioner or it is not the type set when installing
- 1 Unable to open the communication port assigned for this waveform port
- 2 Error setting up the communication port
- 3 Failure during conditioner initialisation
- 4 No conditioner channels detected
- 5 The DLL required is missing or does not have the required entry points (maybe the wrong DLL or out of date)
- 6 There is no device on this channel

See also:

`CondFilter()`, `CondFilterList()`, `CondGain()`, `CondGainList()`, `CondGet()`, `CondOffset()`, `CondOffsetLimit()`, `CondRevision$()`, `CondSet()`, `CondSourceList()`

Cos()

This calculates the cosine of one or an array of angles in radians.

```
Func Cos(x|x[]|{[]...});
```

`x` The angle, expressed in radians, or a real array of angles. The best accuracy of the result is obtained when the angle is in the range -2π to 2π .

Returns When the argument is an array, the function replaces the array with the cosines of all the points and returns either a negative error code or 0 if all was well. When the argument is not an array the function returns the cosine of the angle.

See also:

Abs(), ATan(), Cosh(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan()

Cosh()

This calculates the hyperbolic cosine of one value or an array of values.

```
Func Cosh(x|x[]{|[]...});
```

x The value, or a real array of values.

Returns When the argument is an array, the function replaces each value with its hyperbolic cosine and returns 0. When the argument is not an array the function returns the cosh of the argument.

See also:

Abs(), ATan(), Cos(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sinh(), Sqrt(), Tanh(), Trunc()

Count()

In a time view this counts events and forms the mean level of a waveform channel. It can also be used in a result view to sum bin contents between a start and end bin number.

```
Func Count(chan%, start, finish);
```

chan% The channel number in a time or result view. If the channel does not exist, -1 is returned.

start The start time/bin. If start is greater than finish, the result is 0.

finish The last time/bin. In a time view, if start equals finish, only items that fall exactly at the time count towards the result.

Returns In a time view, for waveform channels it returns the mean waveform level in the time range (gaps in waveforms are ignored). If the time range falls entirely in a gap, the result is 0. For all other channels, it returns the number of events. In a result view it returns the sum of the bins in the range.

If you want to count the number of waveform points between two times (for example where a waveform has gaps), you can use ChanData() to fill an array using code such as:

```
Func CountWave%(chan%, tFrom, tTo)
if ViewKind() then return 0 endif;      'must be time view and wave
if ChanKind(chan%)<>1 and ChanKind(chan%)<>9 then return 0 endif;
const max% := 8000;
var buffer[max%], n% := 0, nGot%;
repeat
  nGot% := ChanData(chan%, buffer, tFrom, tTo, tFrom);
  if (nGot% <= 0) then return n% endif;
  n% += nGot%;
  tFrom += nGot% * BinSize(chan%);
until 0;
end;
```

See also:

ArrSum(), ChanData(), ChanMeasure()

Cursor...()

Cursor()

This returns a cursor position and optionally sets a new position. If you move cursor 0 in a time view, all active cursors 1-9 will search, equivalent to `CursorSearch(1)`.

Func `Cursor(num%{, where});`

`num%` The cursor number to use in the range 1 to 9 (0 to 9 in a time view).

`where` If present, the new cursor position. If this exceeds the x axis range, it is limited to the x axis. In a time view the position is in seconds. In a result view it is a bin number, use `XToBin()` to convert an x axis value to a bin number. In an XY view the position is in x axis units.

Returns The old cursor position or -1 if the cursor doesn't exist.

Examples:

```
Cursor(1,20);           'Set cursor 1 at position 20
where := Cursor(1);    'Get cursor position
```

See also:

`BinToX()`, `XToBin()`, `CursorDelete()`, `CursorLabel()`, `CursorLabelPos()`, `CursorNew()`, `CursorRename()`, `CursorSet()`, `CursorVisible()`

CursorActive()

This function retrieves the current active cursor mode and optionally sets a new mode and search parameters. This is valid for a time view only. The function is equivalent to the vertical cursor mode dialog. Once you have set a cursor mode you can command it to seek with `CursorSearch()` and tell if the search succeeded with `CursorValid()`.

Func `CursorActive(num%);`

Func `CursorActive(num%, mode%{, ch%, str$|minSt, end${, def${, ampLev|n%{, hyPer{, width{, ref${, lev2}}}}}});`

`num%` This is the cursor number, from 0 to 9.

`mode%` If this argument is present, it sets the new cursor mode. Modes in *italics* cannot be used for cursor 0 and are converted to **Static** mode. See the documentation for the cursor mode dialog for details of each mode.

0 Static	7 Rising threshold	14 Data points
1 <i>Maximum value</i>	8 Falling threshold	15 <i>Repolarisation %</i>
2 <i>Minimum value</i>	9 <i>Steepest rising</i>	16 Expression
3 <i>Maximum excursion</i>	10 <i>Steepest falling</i>	17 Turning point
4 Peak find	11 <i>Steepest slope (+/-)</i>	18 <i>Slope%</i>
5 Trough find	12 Slope peak	19 Outside thresholds
6 <i>Threshold</i>	13 Slope trough	20 Inside thresholds

`ch%` The time view channel to search. Use 0 for modes that do not require a channel.

`str$` This is an expression that sets the start time for the search when `num%` is not 0. In expression mode (16) this is the expression to evaluate.

`minSt` This sets the minimum step for cursor 0 in all modes except 0 and 16.

`end$` This string expression sets the end limit of the search. This is ignored for cursor 0 operations when it should be an empty string.

`def$` Optional. If a search fails, and this string evaluates to a valid time, the cursor is positioned at the time and the position is valid. This is ignored for cursor 0 operations when it should be an empty string;

cursor 0 does not move if a search fails. If you do not supply this or it is an empty string, cursors other than cursor 0 are positioned in the middle of the search time range (though this may be modified in future versions to give you more control).

- ampLev** A value or string that sets the amplitude for peaks, threshold level for threshold crossings and baseline level for maximum excursion. It is in the y axis units of the search channel (y axis units per second for slopes). If omitted, the value 0.0 is used. Set it to 0 or an empty string if **ampLev** is not required for the mode.
- n%** The data point count for mode 14. A value of 0 is treated as 1.
- hyPer** The hysteresis for threshold crossings and percent for modes 15 and 18. If omitted, 0 is used. Set it to 0 if **hyPer** is not used by the mode.
- width** This is the width in seconds for all slope measurements. It sets the reference level measurement width in mode 15. It sets the minimum time (Delay) that the data must be above/below a level for threshold measures. It sets the maximum allowed width of a peak or trough. If omitted, 0 is used. Set it to 0 if **width** is not used in the mode or you do not want a time constraint.
- ref\$** This string expression is used in mode 15 to set the time at which the 100% value is measured. The 0% value is measured at the start time.
- lev2** This sets the second threshold level in modes 19 and 20. If omitted, 0 is used. You can supply this as a value or as a string to evaluate.

Returns The active cursor mode at the time of the call.

The arguments **str\$**, **end\$**, **def\$** and **ref\$** are strings holding expressions that evaluate to a time in seconds. They are typically of the form "Cursor(0)+1.3". They can contain any expression that would be valid in the Cursor mode dialog.

See also:

Active cursors, Cursor mode dialog, `CursorActiveGet()`, `CursorNew()`, `CursorSearch()`, `CursorValid()`, `MeasureChan()`, `MeasureX()`

CursorActiveGet()

This gets active cursor parameters set by `CursorActive()` or the cursor mode dialog.

```
Func CursorActiveGet(num%, item% {,&val$});
```

num% This is the cursor number from 0 to 9.

item% This specifies the parameter value as defined for `CursorActive()` to return:

1 ch%	3 end\$	5 ampLev n%	7 width	9 ref\$
2 str\$	4 def\$	6 hyPer	8 minSt	10 lev2

val\$ This optional string variable is updated with the parameter value when **item%** is 2, 3, 4, 5, 9 or 10.

Returns The numerical value of items 1, 5, 6, 7, 8 and 10. Items 5 and 10 may return 0 if the item is an un-parseable string. Otherwise it returns the active cursor mode.

See also:

`CursorActive()`, `CursorExists()`, `CursorNew()`, `CursorSearch()`, `CursorValid()`, `MeasureChan()`, `MeasureX()`

CursorDelete()

Deletes a cursor. It is not an error to delete an unknown cursor (which has no effect). You cannot delete cursor 0; use `CursorVisible(0,0)` to hide cursor 0.

```
Func CursorDelete({num%});
```

num% The cursor number to delete. If omitted, the highest numbered cursor is deleted. Use -1 to delete all cursors 1-9.

Returns The value of `num%` if any cursors were deleted or 0 if no cursor was deleted.

See also:

`Cursor()`, `CursorLabel()`, `CursorLabelPos()`, `CursorNew()`, `CursorRename()`,
`CursorSet()`, `CursorVisible()`

CursorExists()

Use this function to determine if a vertical cursor exists.

```
Func CursorExists(num%);
```

`num%` The cursor number in the range 0-9. Cursor 0 always exists in a time view and never exists in any other view.

Returns 0 if the cursor does not exist, 1 if it does.

See also:

`CursorDelete()`, `CursorNew()`, `CursorValid()`, `HCursorExists()`

CursorLabel()

This gets and optionally sets the cursor label style for the view or for a cursor. You can label cursors with a position and/or the cursor number, or with user-defined text. There are two variants: the first sets styles and labels; the second reads back user-defined labels.

```
Func CursorLabel({style%{, num%{, form$}}});  
Func CursorLabel(&form$, num%);
```

`style%` Set 0=None, 1=Position, 2=Number, 3=Both, 4=User-defined. Omit for no change. Style 4 is used with a format string. Styles 0-3 set the styles of cursors selected by `num%` and the view style for new cursors if `num%` is -1 or omitted. Style 4 is applied to cursors set by `num%`; it does not set the view style.

`num%` A value of -1 or omitting the argument selects all cursors and sets the view style for new cursors, 0-9 selects one cursor.

`form$` A user-defined label string with replaceable fields `%p`, `%n` and `%v(chan)` for position, number and channel value; `chan` is the channel number whose value you require. `%w.dp` and `%w.dv(chan)` formats are allowed where `w` and `d` are numbers that set the field width and number of decimal places.

Returns A cursor style before any change as 0-4 if a single cursor is selected, or the view cursor style as 0-3. If `style%` is omitted or not 0-3, the current view cursor style is not changed.

See also:

`Cursor()`, `CursorDelete()`, `CursorLabelPos()`, `CursorNew()`, `CursorRename()`,
`CursorSet()`

CursorLabelPos()

This lets you set and read the position of the cursor label.

```
Func CursorLabelPos(num% {, pos});
```

`num%` The cursor number. Setting a silly number does nothing and returns -1.

`pos` If present, the command sets the label position as the percentage of the distance from the top of the cursor. Out-of-range values are set to the appropriate limit.

Returns The cursor position before any change was made.

See also:

`Cursor()`, `CursorDelete()`, `CursorLabel()`, `CursorNew()`, `CursorRename()`,
`CursorSet()`

CursorNew()

This command adds a new cursor to the view at the designated position. You cannot use this to create cursor 0 in a time view as this cursor always exists. To show cursor 0 use `CursorVisible(0,1)`. A new cursor is created in Static mode (not active).

Func `CursorNew({where{, num%}});`

where The cursor position. In a time view it is a time in seconds, in a result view, it is the bin number. Use `XToBin()` to convert x axis units to bin numbers. In an XY view it is in x axis units. The position is limited to the x axis range. If the position is omitted, the cursor is placed at a position on the screen that is proportional to the cursor number.

num% If this is omitted, or set to -1, the lowest-numbered free cursor is used. If this is a cursor number, that cursor is created. This must be a legal cursor number or -1.

Returns It returns the cursor number as an integer, or 0 if all cursors are in use.

See also:

`Cursor()`, `CursorActive()`, `CursorDelete()`, `CursorLabel()`, `CursorLabelPos()`, `CursorRenummer()`, `CursorSet()`, `CursorVisible()`, `XToBin()`

CursorOpen()

This command reports on the open state or opens the cursor values and regions dialogs for the current Time or Result view. Dialogs open in the last used position unless it is not usefully visible, when they open centred on the application window. If the dialog is open you can use this command to get the dialog handle and change settings. Use `FileClose()` to close an opened dialog.

Func `CursorOpen({opt%{, mode%{, xZero%{, yZero%|type%}}})`

opt% Set 0 to open the values dialog and 1 to open the regions dialog. Omit or set -1 as the only argument to report on the open state of the cursor dialogs.

mode% Set 1 to open the dialog and show it, 0 or omitted to open and hide it. You may wish to open a window invisible so that you can position it before display.

xZero% This sets the state of the Time Zero or Zero Region check boxes and the associated cursor column selection. Set -2 for unchecked, -1 or omit for no change or set 0 to 9 for the values or 0 to 8 for the regions dialog to check the box and select a column. 0 selects the first column, 1 the second, and so on.

yZero% This sets the state of the Y Zero check box and associated column selection for the value dialog. Set -2 for unchecked, -1 or omit for no change and 0-9 to check the box and select a column. 0 selects the first column, 1 the second, and so on.

type% This sets the measurement type for the regions dialog. Set 1-19 to set a measurement as for `ChanMeasure()`. Omit or set -1 or 0 for no change. See the documentation of the **Cursor Regions** window for details of these measurements.

1 Area	5 Area (scaled)	9 Minimum	13 Abs max.	17 Mean in X
2 Mean	6 Curve area	10 Peak to Peak	14 Peak	18 SD in X
3 Slope	7 Modulus	11 RMS Amplitude	15 Trough	19 Mean of abs
4 Sum	8 Maximum	12 Standard deviation	16 RMS error	

Returns If **opt%** is -1 or omitted the return value is the sum of 1 if the values dialog is open and 2 if the regions dialog is open. Otherwise the return value is the handle of the opened dialog, or 0 if the dialog failed to open for some reason.

See also:

`ChanMeasure()`, `ChanValue()`, `FileClose()`, `Window()`

CursorRenumber()

This command renumbers the cursors from left to right in the view. It has no effect on cursor 0. The visible state of a cursor has no effect on renumbering. Active cursor and label information stays with the cursors, not the number.

```
Func CursorRenumber();
```

Returns The number of cursors that were renumbered (this is the number of vertical cursors that exist, not including cursor 0).

See also:

Cursor(), CursorActive(), CursorDelete(), CursorLabel(), CursorLabelPos(), CursorNew(), CursorSet()

CursorSearch()

This function causes active cursors in a time view to search according to the current cursor mode. You can cause all cursors to search, or a restricted range of cursor numbers. Moving cursor 0 with Cursor(0, newPosition) also causes all cursors 1-9 to search.

```
Func CursorSearch(num% {,stop%});
```

num% This is the first cursor number to run the search defined by the active cursor mode. Set this to 0 to cause cursor 0 to search forwards and to -1 for cursor 0 to search backwards. CursorSearch(0) and CursorSearch(-1) are equivalent to the Ctrl+Shift+Right and Ctrl+Shift+Left key combinations.

stop% This optional argument is the number of the last cursor to try to reposition. If you omit this argument, all cursors from num% upward will search according to their active mode. To reposition a single cursor set stop% the same as num%.

Returns The time that cursor num% moved to or -1 if the search failed. Use CursorValid() to test if searches of other cursors succeeded.

If a search fails, cursor 0 is not repositioned. Other active cursors are set to the middle of the search range (though we reserve the right to modify this in the future, so do not depend on this).

See also:

Cursor(), CursorActiveGet(), CursorNew(), CursorValid(), MeasureChan(), MeasureX()

CursorSet()

This sets the number of vertical cursors in addition to cursor 0. It deletes cursors 1-9, then positions num% cursors equally spaced in the view, numbered in order from left to right. Finally, if any cursors positions are given, they are applied. The cursor labelling style is not changed. This destroys all active cursor information for the deleted cursors.

```
Proc CursorSet(num% {,where1 {,where2 {,where3 {,where4...}}}});
```

num% The number of cursors to display in the range 0 to 9. It is a run-time error to ask for more than 9 or less than 0 cursors. If num% is 0, cursor 0 is hidden.

whereN Optional positions of cursor N (1 to 9). Positions that are out of range are set to the nearest valid position. In a time view positions are in seconds. In a result view, they are the bin number; use XTToBin() to convert x axis units to bin numbers. In an XY view the position is in x axis units. You cannot change the position of cursor 0 in a time view with this command.

Examples:

```
CursorSet(0);           'Delete 1-9, hide cursor 0
CursorSet(2,20,30);    'Delete 1-9, cursor 1 at 20, cursor 2 at 30
```

See also:

BinToX(), Cursor(), CursorDelete(), CursorLabel(), CursorLabelPos(), CursorNew(), CursorReNumber(), CursorVisible(), XToBin()

CursorValid()

Use this function to test if the last search of a cursor in a time view succeeded. Cursor positions are valid if a search succeeds or if the cursor is positioned manually or by a script command. The position of a newly created cursor is valid.

```
Func CursorValid(num%);
```

num% The cursor number to test for a valid search result in the range 0-9.

Returns The result is 1 if the position of the nominated cursor is valid or 0 if it is invalid or the cursor does not exist.

See also:

CursorActiveGet(), CursorNew(), CursorSearch(), CursorVisible(), MeasureChan(), MeasureX()

CursorVisible()

Vertical cursors can be hidden without deleting them. Interactively you can hide cursor 0, but from a script you can show and hide any vertical cursor. Cursors are always made visible by the `Ctrl+n` key combination.

```
Func CursorVisible(num% {,show%});
```

num% The cursor number in the range 0-9 or -1 for all vertical cursors.

show% If present set this to 0 to hide the cursor and non-zero to show it.

Returns The state of the cursor at the time of the call (0=hidden, 1=visible) or -1 if the cursor does not exist. If num% is -1, the result is the number of vertical cursors.

See also:

CursorExists(), CursorNew(), CursorSearch(), CursorValid()

D

Date\$()

This function returns a string holding the date. Use `TimeDate()` to get the date as numbers. For this description, we assume that today's date is Wednesday 1 April 1998, the system language is English and the system date separator is "/". Default argument values are shown **bold**.

```
Func Date$({dayF%{, monF%{, yearF%{, order%{, sep$}}}});
```

dayF% This sets the format of the day field in the date. This can be written as a day of the week or the day number in the month, or both. The options are:

- 1 Show day of week: "Wednesday".
- 2 Show the number of the day in the month with leading zeros: "01".
- 4 Show the day without leading zeros: "1". This overrides option 2.
- 8 Show abbreviated day of week: "Wed".
- 16 Show weekday name first, regardless of the order% field.

Use 0 for no day field. Add the numbers for multiple options. For example, to return "Wed 01", use 11 (1+2+8) as the dayF% argument.

If you add 8 or 16, 1 is added automatically. If you request both the weekday name and the number of

the day, the name appears before the number.

monF% The format of the month field. This can be returned as either a name or a number. If this argument is omitted, the value 3 is used. The options are:

- 0 No month field.
- 1 Show name of the month: "April".
- 2 Show number of month: "04"
- 3 Show an abbreviated name of month: "Apr"
- 4 Show number of month with no leading zeros: "4"

yearF% The format of the year field. This can be returned as a two or four digit year.

- 0 No year is shown
- 1 Year is shown in two digits: "98".
- 2 Year is shown in two digits with an apostrophe before it: "'98".
- 3 Year is shown in four digits: "1998".

order% The order that the day, month and year appear in the string.

- 0 Operating system settings
- 1 month/day/year
- 2 day/month/year
- 3 year/month/day

sep\$ This string appears between the day, month and year fields as a separator. If this string is empty or omitted, Spike2 supplies a separator based on system settings.

For example, `Date$(20, 1, 2, 1, " ")` returns "Wednesday April 1 '98". As 20 is 16+4, we have the day first, even though the `order%` argument places the day in between the month and the year. `Date$()` returns "01/Apr/98".

See also:

`Seconds()`, `FileDate$()`, `TimeDate()`, `Time$()`

Debug()

This command either opens the debug window so you can step through your script, set breakpoints and display and edit variables or it can be used to stop the user entering the debugger with the `Esc` key.

```
Proc Debug({msg$ | Esc%});
```

msg\$ When the command is used with no arguments, or with a string argument, the script stops as though the `Esc` key had been pressed and enters the debugger. If the debugging toolbar was hidden, it becomes visible. If the `msg$` string is present, the string is displayed in the bar at the top of the script window.

Esc% When the command is used with an integer argument, it enables and disables the ability of the user to break out of a running script. If `Esc%` is 0, the user cannot break out of a script into the debugger with the `Esc` key and must wait for it to finish. If `Esc%` is 1, the user can break out. Spike2 enables the `Esc` key each time a script starts, so make this the very first instruction of your script if you want to be certain that the user cannot break out.

This command was included for use in situations such as student use, where it is important that the user cannot break out of a script by accident. It is advisable to test your script carefully before using this option. Once set, you cannot stop a looping script except by forcing a fatal error. Make sure you save your script before setting this option.

See also:

`App(-4)`, `Eval()`, `DebugHeap()`, `DebugList()`, `DebugOpts()`

DebugHeap()

This script command is provided for use by CED engineers to help to debug system problems. It was added at version 6.08. It reports on the state of the application heap, used to dynamically allocate memory. The heap is a list of memory sections. Each section is described by its start address, its size, and if it is in use by the application or is free (available for use). When the application wants more memory, it asks the heap for it. If there is no suitable memory in the heap, the heap requests more memory from the system and uses this to create more heap sections.

The command has the following variants:

Test heap integrity

When called with no arguments, the command tests the integrity of the heap (all command variants do this first). The return values indicate problems in the heap. The error return values apply to all calls to the `Heap()` function.

```
Func DebugHeap( );
```

Returns 0 = heap OK, -1 = `_HEAPBADBEGIN` initial header information is bad or cannot be found, -2 = `_HEAPBADNODE` bad node or the heap is damaged, -3 = `_HEAPBADPOINTER` a pointer into the heap is invalid, -4 = `_HEAPEMPTY` the heap has not been initialised, -5 = unknown error.

Release unused memory

When called with a single argument of -1 (or any value less than 0), the heap will return unused memory back to the operating system. However, I have never seen this make any difference to the data held by the heap.

```
Func DebugHeap(-1);
```

Returns The same values as the call with no arguments.

Set unused heap memory to known value

When called with a single argument that is a positive number, all unused memory that is owned by the heap is set to this value.

```
Func DebugHeap(fill%);
```

`fill%` All unused bytes in the heap are set to the low byte of this value. This can sometimes be useful when you suspect that unused memory is being used by the program.

Returns The same values as the call with no arguments.

Get heap information

The heap is a list of used and unused sections of memory; this call gets information on the number of used and unused sections and the total size of the used sections and the total heap size.

```
Func DebugHeap(info%[{, stats%[]});
```

`info%` An integer array of at least 4 elements. The first four array elements are returned holding:

- 0 The total number of memory sections in the heap.
- 1 The number of used memory sections in the heap.
- 2 The total size of memory controlled by the heap.
- 3 The total size of memory that is in used and controlled by the heap.

`stats%` This optional argument is an integer array of at least 32 elements. The elements are returned with heap information. The n^{th} element is returned holding the number of heap sections of a size between 2^n and $2^{n+1}-1$ bytes. The very first element, which should hold the count of sections that are 1 byte in size actually holds the count of sections that are 0 or 1 byte long.

Returns The same values as the call with no arguments.

Get heap entries

The last call variant returns the entire heap information, and can optionally return the first few bytes of data

held in the heap.

```
Func DebugHeap(walk%[[ ]]);
```

walk% This is an integer matrix with at least 3 columns. Ideally it would have at least as many rows as there are memory sections (both used and free) in the heap. If there are more than 3 columns, the additional columns are returned holding the heap data. Each row of the matrix returns information for one section. The columns hold:

- 0 0 if the section is unused, 1 if it is used.
- 1 The start address of the section.
- 2 The size of the section, in bytes.
- 3 If present, returns the first 4 bytes of data in this section.
- 4 If present, returns the next 4 bytes of data in this section.
- n Returns further data (if it exists) in the section.

Returns If there is no error, the function returns the number of sections in the heap. This can be more than the number of rows in the matrix. If there is an error, the function returns a negative value as described for the call with no arguments.

Example

This example prints out the number of system handles in use and a synopsis of the heap usage. If you suspected that your script was causing a memory leak you could insert this procedure, then call it periodically. If there is a steadily increasing use of handles or heap space, check that you are closing all the views you have created.

```
proc ReportHeap()
var info%[4], stats%[32], i%;
DebugHeap(info%, stats%);
PrintLog("Handles=%d, Heap used=%d kB in %d fragments, %d Kb free in %d fragments\n",
App(-4), info%[3]/1000.0, info%[1], (info%[2]-info%[3])/1000.0, info%[0]-info%[1]);
for i% := 0 to 15 do PrintLog("%6d", pow(2, i%)); next;
PrintLog("\n");
for i% := 0 to 15 do PrintLog("%6d", stats%[i%]); next;
PrintLog("\n");
for i% := 16 to 19 do PrintLog("%4dkB", pow(2, i%)/1024); next;
for i% := 20 to 29 do PrintLog("%4dMB", pow(2, i%)/(1024*1024)); next;
for i% := 30 to 31 do PrintLog("%4dGB", pow(2, i%)/(1024*1024*1024)); next;
PrintLog("\n");
for i% := 16 to 31 do PrintLog("%6d", stats%[i%]); next;
PrintLog("\n");
end;
```

See also:

App(-4), Debug(), Eval(), DebugList(), DebugOpts()

DebugList()

This command is used for debugging problems in the system. It writes information to the Log view about the internal list of “objects” used to implement the script language. Low numbered objects are the integers 0 to 20, higher-numbered items are the instructions that the script compiles to followed by objects that represent the built-in script commands.

```
Proc DebugList(list% {, opt%});
```

list% This determines what to list and also controls the accumulation of timing information.

- 3 Disable the accumulation of timing information (the normal state) for calls to built-in functions.
- 2 Enable the accumulation of timing information. This is normally used as a diagnostic of slow script performance when we need to figure out where the time is being spent.
- 1 Reset accumulated times and call counts to 0.
- 0 List a summary of the DebugList() options in the Log view.

- 1 List the names of fixed objects (constants and operators). This is usually only of interest to CED programmers.
- 2 List the names of permanent objects (constants, operators and built-in commands).
- 3 List built-in commands (things like `NextTime()`).
- >3 List information for the object with the index `list%`.

`opt%` This optional argument (default value 0) sets the additional object information to list and is the sum of: 1= list the index number, 2= list the type, 4= list timing information for the built-in script commands. The timing information is three numbers: the number of times the command was called, the total time in seconds used and the time per call in microseconds.

To use the timing information:

```
DebugList(-3); 'Disable timing
DebugList(-1); 'Reset call counts and times
DebugList(-2); 'Enable timing
... your code to be timed
DebugList(-3); 'stop accumulating times
PrintLog("Command name      Total calls      Seconds      us/call Index\n");
DebugList(3, 5); 'list times and index numbers
```

Here is some typical output. The timing is from the point where the command arguments have been prepared up to the point where the command returns control to the script language. These times were measured on an AMD Athlon 1900+ rated processor.

Command name	Total calls	Seconds	us/call	Index
...				
Asc	537	0.000512	0.953	530
Chr\$	668	0.001169	1.751	531
DelStr\$	13	0.000030	2.299	532
InStr	1076	0.001263	1.174	533
LCase\$	130	0.000230	1.769	534
Left\$	334	0.000742	2.221	535
Mid\$	3751	0.005480	1.461	536
Right\$	79	0.000155	1.959	537
...				

The timing information is normally used by CED programmers to check that functions are working with a reasonable efficiency. If you should discover that a particular function is using a lot of time, you may be able to optimise your use of the function to improve matters. If you think that a particular function is slow, let us know; we may be able to improve it.

See also:

`App(-4)`, `Eval()`, `Debug()`, `DebugHeap()`, `DebugOpts()`

DebugOpts()

This command is used for debugging problems in the system. It controls internal options used for debugging at the system level.

```
Func DebugOpts(opt% {,val%});
```

`opt%` This selects the option to return (and optionally to change). A value of 0 prints a synopsis of available options to the Log view and the current value of each option. Values greater than 0 return the value of that option, and print the option information to the Log view. At the time of writing, only option 1, dump compiled script to the file `default.cod` is implemented.

`val%` If present, this sets the new value of the option.

See also:

`App(-4)`, `Eval()`, `Debug()`, `DebugHeap()`, `DebugList()`

DelStr\$()

This function removes a sub-string from a string.

```
Func DelStr$(text$, index%, count%);
```

text\$ The string to remove characters from. This string is not changed.

index% The start point for the deletion. The first character is index 1. If this is greater than the length of the string, no characters are deleted.

count% The number of characters to remove. If this would extend beyond the end of the string, the remainder of the string is removed.

Returns The original string with the indicated section deleted.

See also:

Asc(), Chr\$(), InStr(), LCase\$(), Left\$(), Len(), Mid\$(), Print\$(), ReadStr(), Right\$(), Str\$(), UCase\$(), Val()

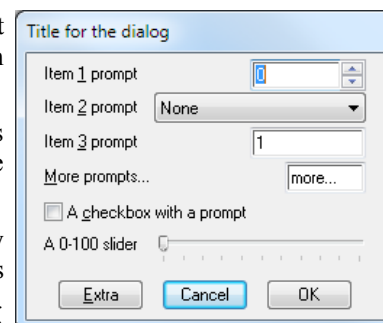
Dialogs

You can define your own dialogs to get information from the user. You can define dialogs in a simple way, where each item of information has a prompt, and the dialog is laid out automatically, or you can build a dialog by specifying the position of every item. A simple dialog has the structure shown in the diagram:

The dialog is arranged in terms of items. Unless you specifically request otherwise, the dialog items are stacked vertically above each other with buttons arranged at the bottom.

The dialog has a title of your choosing at the top and **OK** and **Cancel** plus user-defined buttons at the bottom. When the dialog is used, pressing the **Enter** key is equivalent to clicking on **OK**.

This form of dialog is very easy to program; there is no need to specify any position or size information, the system works it all out. Some users require more complicated dialogs, with control over the field positions. This is also possible, but harder to program. You are allowed up to 1000 fields in a dialog.



In more complex cases, you specify the position (and usually the width) of the box used for user input. This allows you to arrange data items in the dialog in any way you choose. It requires more work as you must calculate the positions of all the items.

Steps to program a dialog

Generating and using a dialog is a three step process:

1. Use `DlgCreate()` to start designing a dialog. The dialog starts with an **OK** and a **Cancel** button, but you can change that in step 2.
2. Add fields to the dialog with commands like `DlgInteger()`, `DlgChan()` and so on. You can add and modify buttons with `DlgButton()`. You can also configure how the dialog behaves with `DlgAllow()` and set mouse interactions with `DlgMouse()`.
3. Use `DlgShow()` with a list of arguments, one per dialog field, to get and set the editable field contents. As there is a limit on the number of arguments allowed in the script language, you can also use arrays as arguments; an array with *n* elements matches *n* fields in the dialog.

Version 5 extensions

From version 5 onwards you can add and manipulate buttons, collect a time or x axis value and add a group box. You can define script functions that are called in response to button presses and user changes to the dialog and an idle-time function that is called repeatedly whilst the dialog is waiting for user actions. All these functions can enable and disable, hide and show and modify dialog items. `DlgChan()` can now get a channel

from an XY view. You can add tooltips for all the prompts in a dialog to give users a more detailed explanation of a field. Finally, there are extensions to the integer, real and string fields that allow you to define a drop-down list of selectable items to copy into the fields and you can add a spin control to both integer and real numeric fields. All dialogs created in previous versions of Spike2 should work without any change.

Version 6 extension

We added `DlgGetPos()` at version 6.06 to get the dialog position. Most users like to choose where a dialog is positioned; this lets you find out where it has been placed and restore it the next time.

Version 7 extension

The `DlgMouse()` function was extended at version 7.01 to link mouse movements and button clicks to user-defined functions while your dialog is active in exactly the same way as for the `ToolbarMouse()` command.

The `DlgSlider()` function was added at version 7.07 plus the ability for the x positions of dialog items to be set negative, meaning position relative the the right-hand edge of the dialog.

Dialog units

Positions within a dialog are set in dialog units. In the x (horizontal) direction, these are in multiples of the maximum width of the characters '0' to '9'. In the y (vertical) direction, these are in multiples of the line spacing used for simple dialogs. The maximum y position is 40. You are allowed to use negative x positions, meaning place the right hand edge of the item this many dialog units in from the dialog right edge. There is a 2 dialog unit border to left and right of the dialog, so the standard x position (if x is omitted or 0) is 2. and the standard indent from the right is -2. Unless you intend to produce complex dialogs with user-defined positions, you need not be concerned with dialog units at all.

Simple dialog example

The simple example dialog shown above can be created by this code:

```
var ok%, item1%, item2%, item3, item4$:= "more...", item5, item6;
DlgCreate ("Title for the dialog");      'start new dialog
DlgInteger(1, "Item &1 prompt", 0, 10, 0, 0, 1); 'range 0-10, spinner
DlgChan   (2, "Item &2 prompt|Tip", 1);    'Waveform channel list
DlgReal   (3, "Item &3 prompt", 1.0, 5.0); 'real, range 1.0-5.0
DlgString (4, "&More prompts...|Tip", 6);  'string, any characters
DlgCheck  (5, "A &check box with a prompt"); 'a check box item
DlgButton (2, "&Extra||Tooltip");         'button 2, tooltip
DlgSlider (6, "A 0-100 slider", 0, 100, 10, 5); 'a slider with ticks

ok% := DlgShow(item1%, item2%, item3, item4$, item5, item6); 'show dialog
```

Prompts, & and tooltips

In the functions that set an item with a prompt, if you precede a character in the prompt with an ampersand (&), the following character is underlined and is used by Windows as a short-cut key to move to the field or activate the button. All static dialog items except a group box allow you define a tooltip by appending a vertical bar followed by the tooltip text to the `text$` argument. For example:

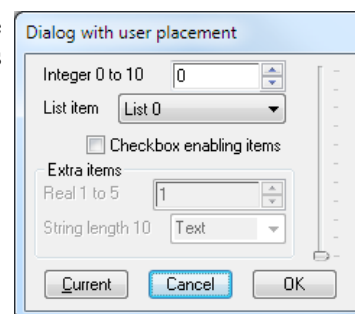
```
DlgReal(3, "Rate|Enter the sample rate in Hz", 100, 500);
```

Buttons allow you to specify an additional activation key and an optional tooltip by adding a vertical bar followed by the key code and then another vertical bar followed by the tooltip. See the `label$` argument of `ToolbarSet()` for details of key codes.

More complex example

This example shows how to respond to user actions within a dialog. We use a check box to enable and disable a group of items and a button that displays the current values of dialog items. The numbered fields are:

- 1 An integer, range 0-10 with a spinner
- 2 A drop list of 4 items
- 3 A check box, used to enable items 4 and 5
- 4 A real number with a spinner
- 5 A string with a drop down list of items
- 6 A vertical slider positioned on the right



We have added button 2 (buttons 0 and 1 are Cancel and OK) and a group box around items 4 and 5. To make room for the group box, the y positions of items 4 and 5 are set explicitly. To make room for the slider, we have defined a negative x offset and applied it to all the non-button fields.

With `DlgAllow()` we have set `Func Change%(item%)` to be called whenever the user changes a selection or check box or when an editable field loses the input focus. The `item%` argument is set to the item number that changed or to 0 if the dialog is appearing for the first time. We are interested in item 3, the check box, and we use the state to enable or disable the group box and the items inside it.

`Func Current%()` is linked to the "Current" button and can be activated with the C and F1 keys and has a tooltip. The function displays a message box that lists the current values of items in the dialog.

```
var ok%, item1%, item2%, item3%, item4, item5$:= "Text", item6, gp%;
var xo := -8; 'Space from right edge for slider
DlgCreate("Dialog with user placement",0,0,40,7.5);
DlgInteger(1,"Integer 0 to 10",0,10,xo,1,1); 'Int with spinner
DlgList(2,"List item","List 0|List 1|List 2|List 3", 4, xo, 2);
DlgCheck(3, "Check box enabling items",xo,3); 'check box item
DlgReal(4, "Real 1 to 5",1.0,5.0,xo,4.5,0.5); 'Real with spinner
DlgString(5, "String length 10",10,"",xo,5.5, 'String item with
"String 1|String 2|String 3"); 'drop-down list
DlgSlider(6, -5.8, 0, 100, 10, 1, -2, 1); 'rhs vert slider
DlgButton(2,"&Current|Ox70|Tooltip", Current%);'button F1+function
DlgAllow(0x3ff, 0, Change%); 'Allow all, no idle, change function
gp% := DlgGroup("Extra items",1,3.8,xo+1,2.9); 'Group box
DlgMouse(-1, -1, 1, 1, MouseDown%, MouseUp%, MouseMove%);
ok% := DlgShow(item1%,item2%,item3%,item4,item5$,item6);
Halt;

Func Change%(item%)
var v%;
docase
case ((item% = 3) or (item% = 0)) then '0 is initial setup
v% := DlgValue(3); 'get check box state
DlgEnable(v%, gp%, 4, 5); 'enable groupbox+items 4, 5
endcase;
return 1; 'Return 1 to keep dialog running
end;

Func Current%()
var v1%, v2%, v3%, v4, v5$;
v1% := DlgValue(1); v2% := DlgValue(2); 'Retrieve values
v3% := DlgValue(3); v4 := DlgValue(4); v5$ := DlgValue$(5);
Message("Values are %d, %d, %d, %g and %s",v1%,v2%,v3%,v4,v5$);
return 1; 'Return 1 to keep the dialog running
end;

func MouseDown%(vh%, chan%, x, y, flags%)
PrintLog("Down: chan=%2d, x=%g, y= %g, flags% = %d\n", chan%, x, y, flags%);
return 1024; 'measurement cursor + a line linking start to end
end;

func MouseMove%(vh%, chan%, x, y, flags%)
PrintLog("Move: chan=%2d, x=%g, y= %g, flags% = %d\n", chan%, x, y, flags%);
return 0; 'keep same cursor as for the mouse down
end;
```

```
func MouseUp%(vh%, chan%, x, y, flags%)
PrintLog("Up: chan=%2d, x=%g, y= %g, flags% = %d\n", chan%, x, y, flags%);
return 1;      'do not close the toolbar
end;
```

DlgAllow()

Call this function after `DlgCreate()` and before `DlgShow()` ends to enable dialog idle time processing, advanced call-back features and dynamic access to the dialog fields. There are no restrictions on what call-back functions can do. However, it is not sensible to place time-consuming code in an idle call-back function or to do anything other than check dialog fields and possibly display a warning message in a dialog-item-change function. Call-back functions use `DlgValue()`, `DlgEnable()` and `DlgVisible()` to manipulate the dialog fields. `DlgAllow()` cannot be used during `DlgShow()` before version 6.11. If you open a dialog within a dialog, this commands applies to the currently open dialog until you call `DlgCreate()`, after which it applied to the new dialog until you return from `DlgShow()`.

```
Proc DlgAllow(allow% {,func id%(){, func ch%()}});
```

allow% A number that specifies the actions that the user can and cannot take while interacting with Spike2. See `Interact()` for a full description.

id%() This is an integer function with no arguments. Use the name with no brackets, for example `DlgAllow(0,Idle%)`; where `Func Idle%()` is a script function. When `DlgShow()` executes, the function is called repeatedly in system idle time, as for the `ToolbarSet()` idle function unless a change function, a button function or a Mouse function is active.

If the function return value is greater than 0, the dialog remains open. A zero or negative return value closes the dialog and `DlgShow()` returns the same value.

If this argument is omitted or 0, there is no idle time function.

ch%() This is an integer function with one integer argument, for example `Func Changed%(item%)`. You would use `DlgAllow(0,0,Changed%)`; to link this function to a dialog. Each time the user changes a dialog item, Spike2 calls the function with the argument set to the changed item number. There is an initial call with the argument set to 0 when the dialog is about to be displayed.

A field is deemed to change when the user clicks a check box or changes a selection in a list or moves the focus from an editable item after changing the text. For real and integer values, the new value must be in range.

If the change function returns greater than 0, the change is accepted. If the return value is zero, the change is resisted and the focus set back to the changed item. If the return value is negative, the dialog closes and `DlgShow()` returns this value and the arguments are not updated.

You can find an example above (more complex example).

DlgButton()

Dialogs created by `DlgCreate()` have **Cancel** and **OK** buttons; this function adds, deletes and changes buttons. You can link a script function to a button and use the function return value to decide if the dialog should close. Use this function after `DlgCreate()` and before `DlgShow()`. To change a button label after you call `DlgShow()`, use `DlgValue$()` from a dialog call-back function. You can also call this function with no arguments to get the number of the last button that was pressed (added at version 7.01). There are two command variants:

```
Func DlgButton(but%, text${, func ff% {, x, y}});
Func DlgButton();
```

but% The button number from 0 to 200. Button 0 is the cancel button, 1 is the OK button. Button numbers higher than 1 create new buttons.

text\$ This sets the button label. Set an empty string to delete a button. You cannot delete button 1; the label is set back to OK if you try. The label text can be followed by an optional key code and an optional tooltip separated by vertical bars. See the `label$` argument of `ToolbarSet()` for details of the format.

If you set a key code, the button can be activated even when the dialog does not have the input focus as long as it is the topmost user dialog and you have not created a toolbar or interact bar from a function linked to the dialog. This allows you to drag cursors in a window, then use the key code without the need to click in the dialog to activate it first.

`ff%` This is an integer function with no arguments that is called when the button is used. You must supply the function name only; do not include the left and right round brackets after it or the function will be evaluated during the `DlgButton()` call and no function will be linked to the button. Set the argument to zero or omit it if you don't want a button function, in which case clicking the button closes the dialog, `DlgShow()` returns the button number and the `DlgShow()` arguments are updated for all buttons except 0.

If you supply a function, it is called each time the button is used and the function return value determines what happens next:

<0 The button acts as **Cancel**. The dialog closes, `DlgShow()` returns this value and its arguments are not updated.

0 The button acts as **OK**. The dialog closes and the `DlgShow()` return value is the button number and its arguments are updated.

>0 The dialog continues to display.

The button function can use `DlgEnable()`, `DlgValue()` and `DlgVisible()` and can also create and show subsidiary dialogs.

`x,y` Set the button position in dialog units, both or neither of these must be supplied. If the button position is not supplied it will be positioned at the bottom of the dialog.

Returns The call with no arguments returns the number of the last button that was pressed. This allows you to service multiple buttons with the same user-defined button. The call that creates buttons returns 0.

Example

```
func DoButton%()
Message("Button %d", DlgButton()); 'report button number
return 1; ' leave dialog open
end;

DlgCreate("Example of button replacement");
DlgText("This dialog demonstrates using a button", 0, 1);
DlgButton(0, ""); ' remove the Cancel button
DlgButton(2, "My &Button|Press to say hello", DoButton%);
DlgShow(); ' no arguments as no variable fields defined
```

There is a simple example here. You can find a more complex example here.

DlgChan()

This function defines a dialog entry that lists channels that meet a specification for time, result and XY views. For simple dialogs, the `wide`, `x` and `y` arguments are not used. Channel lists are checked or created when the `DlgShow()` function runs. If the current view is not a time, result or XY view, the list is empty.

```
Proc DlgChan(item%, text$|wide, mask%|list%[]{, x{, y}});
```

`item%` This sets the item number in the dialog in the range 1 to the number of items.

`text$` The prompt to display, optionally followed by a vertical bar and tooltip text.

`wide` This is an alternative to the prompt. It sets the width in dialog units of the box used to select a channel. If `wide` is omitted the number entry box has a default width of the longest channel name in the list or 12, whichever is the smaller.

`mask%` This determines the channels to display. Select channel types by adding the following codes, which are given as decimal and hexadecimal.

1 0x1	Waveform or result view channel
2 0x2	Event+ and Event- channels
4 0x4	Event +- channels (level data)
8 0x8	Marker channels

16 0x10	WaveMark data
32 0x20	TextMark data
64 0x40	RealMark data
128 0x80	Unused/deleted disk channels
256 0x100	Deleted channels on disk
512 0x200	Real wave channel

If none of the above values are used or this is an XY view, the list includes all channels. Add the following codes to exclude channels from the list:

1024 0x400	Exclude visible channels
2048 0x800	Exclude hidden channels
4096 0x1000	Exclude time view disk channels but not their duplicates
8192 0x2000	Exclude memory channels but not their duplicates
16384 0x4000	Exclude duplicated channels
32768 0x8000	Exclude selected channels
65536 0x10000	Exclude non-selected channels
2097152 0x200000	Exclude virtual channels but not their duplicates

Finally, the following add special entries to the list and control the display:

131072 0x20000	Add None as an entry in the list, returns 0
262144 0x40000	Add All channels as an entry in the list, returns -1
524288 0x80000	Add All visible channels as an entry, returns -2
1048576 0x100000	Add Selected as an entry in the list, returns -3
4194304 0x400000	No channel type or shorten long XY channel titles

`list%` As an alternative to a mask, you can pass in a channel list (as constructed by `ChanList()`). This must be an array of channel numbers, with the first element of the array holding the number of channels in the list. From version 6.03 you can add the special entries into the first element. For example, if you have 5 channels following and you want to add **None** as an option, use `list%[0]:=0x20005`.

`x` If omitted or zero, the selection box is right justified in the dialog box, otherwise positive values set the position of the left end of the channel selection box in dialog units and negative values set the position of the right hand end.

`y` If omitted or zero, this takes the value of `item%`. It is the position of the channel selection box in dialog units.

The variable passed to `DlgShow()` for this field should be an integer. If the variable passed in holds a channel number in the list, the field shows that channel, otherwise it shows the first channel in the list (usually **None**). The result from this field in `DlgShow()` is a channel number, or 0 if **None** is selected, -1 if **All channels** is selected, -2 if **All visible channels** is selected or -3 if **Selected** is chosen.

There is a simple example here.

DlgCheck()

This defines a dialog item that is a check box (on the left) with a text string to its right. For simple dialogs, the `x` and `y` arguments are not used.

```
Proc DlgCheck(item%, text${, x{, y}});
```

`item%` This sets the item number in the dialog in the range 1 to the number of items.

`text$` The prompt to display, optionally followed by a vertical bar and tooltip text.

`x, y` The position of the bottom left hand corner of the check box in dialog units. If omitted, `x` is set to 2 and `y` to `item%`. When used without these fields, this behaves exactly like the simple dialog functions, and can be mixed with them.

The associated `DlgShow()` variable should be an integer. It sets the initial state (0 for unchecked, not 0 for checked) and returns the result as 0 (unchecked) or 1 (checked). This item does not have a prompt. If you use

`DlgValue()`, a string value refers to the text and a numeric value refers to the check box state.

There is a simple example here.

DlgCreate()

This function starts the definition of a dialog and clears any `DlgAllow()` settings. It also kills off any previous dialog that might be partially defined. The `scr%` and `rel%` arguments were added at version 6.03.

```
Func DlgCreate(title$,x{,y{,wide{,high{,help{,scr%{,rel%}}}}}});
```

`title$` A string holding the title for the dialog.

`x,y` Optional, taken as 0 if omitted. The position of the top left hand corner of the dialog as a percentage of the screen size. The value 0 means centre the dialog. Values out of the range 0 to 95 are limited to the range 0 to 95.

`wide` The width of the dialog in dialog units. If this is omitted, or set to 0, Spike2 works out the width for itself, based on the items in the dialog. If this is negative, `-wide` is the limit for the dialog width (before Spike2 version 8, negative widths were treated as 0). In any case, the dialog is limited to the main screen width.

`high` The height of the dialog in dialog units. If omitted, or set to 0, Spike2 works it out for itself, based on the dialog contents. If this is negative, `-high` is the limit for the dialog height (before Spike2 version 8, negative heights were treated as 0). In any case, the dialog is limited to the main screen height.

`help` A string or numeric identifier that identifies the help page to be displayed if the user requests help when the dialog is displayed. Use -1 for no defined help page. This is unlikely to be useful unless you create your own Help file and then use the `Help()` script command to change the default help file.

`scr%` Optional screen selector for views, dialogs and the application window. If omitted or -1, positions are relative to the application window. Otherwise, 0 selects the entire desktop rectangle and greater values select a particular screen rectangle (but see `rel%`). See `System()` for more screen information.

`rel%` Ignored unless `scr%` is 0 or greater. Set 0 or omit for positions relative to the intersection of the rectangle set by `scr%` and the application window, 1 for positions relative to the `scr%` rectangle. If there is no intersection, there is no position change.

Returns This function returns 0 if all was well, or a negative error code.

For simple use, only the first argument is needed. The remainder are for use with more complicated menus where precise control over menu items is required.

Maximum dialog size

In Spike2 version 8, the dialog size is limited to the size of the primary monitor attached to the system. Previously, the maximum size had an arbitrary limit of 180x40 in dialog units. If either dimension of the primary screen is less than the previous limits, we allow the old limits for reasons of backwards compatibility.

Use of & in prompts

In the functions that set an item with a prompt, if you precede a character in the prompt with an ampersand "&", the following character is used by Windows as a short-cut key to move to the field and the character is underlined. Ampersand characters are ignored on systems that do not use this mechanism.

DlgEnable()

Use this only from a dialog call-back function to enable or disable dialog items. With one argument, it returns the enabled state of an item; with two or more arguments it sets the enabled state of one or more items. Prompts and spin controls associated with the item are also enabled or disabled. `DlgEnable()` was new in version 5.

```
Func DlgEnable(en%, item%|item%[] {, item%|item%[]...});  
Func DlgEnable(item%);
```

`en%` Set 0 to disable list items, 1 to enable them and 2 to enable and give the first item the input focus.

Input focus changes should be used sparingly to avoid user confusion; they can cause button clicks to be missed.

`item%` An item number or an array of item numbers of dialog elements. The item number is either the number you set, or the number returned by `DlgText()` or `DlgGroup()`, or `-button`, where `button` is the button number. You cannot access prompts separately from their items as this makes no sense.

Returns When called with a single argument it returns the enabled state of the item, otherwise it returns 0.

You can find an example here.

DlgFont()

Before Spike2 version 8, the font used in user-defined dialogs and for the `Input()`, `Input$()` and `Query()` commands was the system `ANSI_VAR_FONT`, which is usually MS Sans Serif 8pt. This is a proportionally spaced raster font, which does not look as nice as a TrueType or ClearType font as it does not take advantage of the ability to use fractional pixel positions on modern displays. However, by sticking with this font, user-defined dialogs look the same and layout the same across all windows platforms.

The `DlgFont()` script command gives you two more choices of font for user-defined dialogs, but with the penalty that you cannot be certain that complex user-defined dialogs will layout in the same way in different versions of Windows or with different user options. The font used by a user-defined dialog is set when the `DlgShow()` command is executed, so `DlgFont()` must be run before this. Each time a script starts the default font is reset, equivalent to `DlgFont(0)`. It is possible that in the future we will provide a Preferences option to change the default.

Func DlgFont({font%})

`font%` An optional argument to set the desired font policy. If omitted, no change is made. There are three options:

- 0 This selects the original `ANSI_VAR_FONT`. This is automatically set each time you run a script.
- 1 This sets the same font that Spike2 uses for dialogs (such as the About Spike2 dialog). This is usually an 8 pt TrueType font (often Tahoma), so will look sharper than option 0 and will usually have a similar spacing.
- 2 This sets the font that Windows is using for system dialogs.

Returns The font code that was in use at the time of the call (0, 1 or 2).

Setting option 1 will usually not cause too much difference in layout and the result should look better than option 0. Setting option 2 is more problematic as the default since Vista is Segoe UI 9 pt, and this is likely to cause layout incompatibilities, especially if you make complex dialogs.

See also:

`DlgShow()`, `Input()`, `Input$()`, `Query()`

DlgGetPos()

This can only be used from a dialog call-back function to get the position of the top left corner of the dialog. The call-back functions are set by `DlgAllow()` and `DlgButton()`. To get the final dialog position you need to have call-backs for any button press that would close the dialog, or in an idle routine. This function was added at Spike2 version 6.06.

Func DlgGetPos(&x, &y{, scr%{, rel%}});

`x,y` Returned holding the position of the top left hand corner of the dialog relative to the rectangle defined by `scr%` and `rel%`.

`scr%` Optional screen selector for views, dialogs and the application window. If omitted or -1, positions are relative to the application window. Otherwise, 0 selects the entire desktop rectangle and greater values select a particular screen rectangle (but see `rel%`). See `System()` for more screen information.

`rel%` Ignored unless `scr%` is 0 or greater. Set 0 or omit for positions relative to the intersection of the rectangle set by `scr%` and the application window, 1 for positions relative to the `scr%` rectangle. If there is no intersection, there is no position change.

Returns 1 if the position was returned, or -1 if the rectangle set by `scr%` and `rel%` is of zero size.

DlgGroup()

This routine creates a group box, which is a rectangular frame with a text label at the top left corner. You can use this between calls to `DlgCreate()` and `DlgShow()`. There is nothing for the user to edit in this item, so you do not supply an item number and there is no matching argument in `DlgShow()`. However, the returned number is an item number (above the values used to match items to `DlgShow()` arguments) that you can use in call-back functions to identify the group box. This function was added at version 5.

```
Func DlgGroup(text$, x, y, width, height);
```

`text$` The text to display at the top left of the group box. Any tooltip text is ignored.

`x,y` The position of the top left corner of the group box.

`width` If positive, the width of the group box in dialog units. If negative, this is the offset of the right hand side of the group box from the right hand edge of the dialog.

`height` The height of the group box in dialog units.

Returns The routine returns an item number so that you can refer to this in call-back functions to use `DlgVisible()` and `DlgEnable()`.

You can find an example here.

DlgInteger()

This function defines a dialog entry that edits an integer with an optional spin control or drop down list of selectable items. The numbers you enter may not contain a decimal point. For simple dialogs, the `wide`, `x`, `y`, `sp%` and `li$` arguments are not used. The `sp` and `li$` arguments were new in version 5.

```
Proc DlgInteger(item%, text$|wide, lo%, hi%{, x{, y{, sp%|li$}}});
```

`item%` This sets the item number in the dialog in the range 1 to the number of items.

`text$` The prompt to display, optionally followed by a vertical bar and tooltip text.

`wide` This is an alternative to the prompt. It sets the width in dialog units of the box in which the user types the integer. If the width is not given the number entry box has a default width of 11 digits (or width needed for the number range?).

`lo%` The start of the range of acceptable numbers.

`hi%` The end of the range of acceptable numbers.

`x` If omitted or zero, the number entry box is right justified in the dialog. Otherwise positive values set the position of the left side of the box relative to the left-hand side of the dialog in dialog units and and negative values set the right side position relative to the right-hand side of the dialog.

`y` If omitted or zero, this takes the value of `item%`. It is the position of the bottom of the number entry box in dialog units.

`sp%` If present and non-zero, this adds a spin box with a click increment of `sp%`. You can change this value dynamically with `DlgValue()`.

`li$` If present, this argument is a list of items separated by vertical bars that can be selected into the integer field, for example "1|10|100".

The variable passed into `DlgShow()` as argument number `item%` should be an integer. The field starts with the value of the variable if it is in the range. Otherwise, it is limited to the nearer end of the range. You can find an example of `DlgInteger()` use here.

DlgLabel()

This function sets an item with no editable part that is used as a label. For simple dialogs, the `wide`, `x` and `y` arguments are not used. You can add text to a dialog without using an item number with `DlgText()`.

```
Proc DlgLabel(item%, text${, x{, y}});
```

- `item%` This sets the item number in the dialog in the range 1 to the number of items.
- `text$` The prompt to display, optionally followed by a vertical bar and tooltip text.
- `x` If omitted or zero, the text is left justified in the dialog. Otherwise positive values set the position of the left side of the text relative to the left-hand side of the dialog in dialog units and and negative values set the right side position relative to the right-hand side of the dialog.
- `y` If omitted or zero, this takes the value of `item%`. It is the position of the bottom of the text in the dialog in dialog units.

When you call `DlgShow()`, you must provide a dummy variable for this field. The variable is not changed and can be of any type, but must be present.

DlgList()

This defines a dialog item for a one of `n` selection. Each of the possible items to select is identified by a string. For simple dialogs, the `wide`, `x` and `y` arguments are not used.

```
Proc DlgList(item%, text$|wide, list${[]|list${, n%{, x{, y}}});
```

- `item%` This sets the item number in the dialog in the range 1 to the number of items.
- `text$` The prompt to display, optionally followed by a vertical bar and tooltip text. The list box width is the smaller of the longest string in the list or 20.
- `wide` This is an alternative to the prompt. It sets the width of the box in which the user selects an item. If you need a prompt you can use `DlgText()`.
- `list$` A string of list items separated by a “|” character or an array of strings, one per item. Long strings are truncated. The “|” method was new in version 3.
- `n%` The number of entries to display. If this is omitted, or if it is larger than the number of entries provided, then all the entries are displayed.
- `x` If omitted or zero, the number list box is right justified in the dialog. Otherwise positive values set the position of the left side of the box relative to the left-hand side of the dialog in dialog units and and negative values set the right side position relative to the right-hand side of the dialog.
- `y` The selection box vertical position in dialog units. If omitted, the value of `item%` is used.

The result obtained from this is the index into the list of the list element chosen. The first element is number 0. The variable passed to `DlgShow()` for this item should be an integer. If the value of the variable is in the range 0 to `n-1`, this sets the item to be displayed. Otherwise, the first item in the list is displayed.

The following example shows how to set a list:

```
var ok%, which%:=0;           'string list, test for OK, result
DlgCreate("List example");   'Start the dialog
var list${3};                'these strings are the choices
list${0} := "one"; list${1} := "two"; list${2} := "three";
DlgList(1, "Choose", list${}); 'Add the list to the dialog
ok% := DlgShow(which%);     'Display dialog, wait for user
```

From version 3, you can replace the third, fourth and fifth lines with:

```
DlgList(1, "Choose", "one|two|three"); 'version 3 onwards
```

DlgMouse()

Use after `DlgCreate()` and before `DlgShow()`. There are two variants of the command; the first sets the initial mouse pointer position when the dialog opens. If you do not use this command variant, the mouse pointer is not moved when the dialog opens. This command can be particularly useful if you are using multiple screens.

```
Proc DlgMouse(item%);
```

`item%` The item number of an element of the dialog at which to position the mouse pointer when the dialog opens. This is either the number you set for the dialog item, or the number returned by `DlgText()` or `DlgGroup()`, or `-button`, where `button` is the button number.

The second variant gives you access to the mouse positions and left button mouse clicks in Time, Result and XY views when the mouse is over a data channel while a dialog is active. This functions in exactly the same way as the `ToolbarMouse()` command, so see there for full details and a description of the mouse down, up and move functions. The command arguments are:

```
Proc DlgMouse(vh%, ch%, mask%, want%, Func Down%{, Func Up%{, Func Move%});
```

DlgReal()

This function defines a dialog entry that edits a real number. For simple dialogs, the `wide`, `x` and `y` arguments are not used. The `sp` and `li$` arguments were new in version 5.

```
Proc DlgReal(item%, text$|wide, lo, hi{, x{, y{, sp|li${, pre%}}});
```

`item%` This sets the item number in the dialog in the range 1 to the number of items.

`text$` The prompt to display, optionally followed by a vertical bar and tooltip text.

`wide` This is an alternative to the prompt. It sets the width in dialog units of the box in which the user types a real number. If `wide` is not given the box has a default width of 12 digits.

`lo, hi` The range of acceptable numbers.

`x` If omitted or zero, the number entry box is right justified in the dialog. Otherwise positive values set the position of the left side of the box relative to the left-hand side of the dialog in dialog units and and negative values set the right side position relative to the right-hand side of the dialog.

`y` The number edit box vertical position in dialog units. If omitted, the value of `item%` is used.

`sp` If present and non-zero, this adds a spin box with a click increment of `sp`. You can change this value dynamically with `DlgValue()`.

`li$` If present, this argument is a list of items separated by vertical bars that can be selected into the editing field, for example "1.0|10.0|100.0".

`pre%` If present, this sets the number of significant figures to use to represent the number in the range 6 (the default) to 15.

The variable passed into `DlgShow()` should be a real number. The field will start with the value of the variable if it is in the range, otherwise the value is limited to `lo` or `hi`.

There is a simple example here.

DlgShow()

This function displays the dialog you have built and returns values from the fields identified by item numbers, or makes no changes if the dialog is cancelled. Once the dialog has closed, all information about it is lost. You must create a new dialog before you can use this function again.

```
Func DlgShow(&item1|item1[], &item2|item2[], &item3|item3[] ...);
```

item For each dialog item with an item number, you must provide a variable of a suitable type to hold the result. It is an error to use the wrong variable type, except an integer field can have a real or an integer variable. Items created with `DlgLabel()` must have a variable too, even though it is not changed.

The variables also set the initial values. If an initial value is out of range, the value is changed to the nearest legal value. In the case of a string, illegal characters are deleted before display.

In addition to passing a simple variable, you can pass an array. An array with *n* elements matches *n* items in the dialog; this allows you to have many more items in a dialog than the argument limit for a function call. The array type must match the items.

Returns 0 if the user clicked on the **Cancel** button, or 1 if the user clicked on **OK**. You can also add your own buttons with the `DlgButton()` command and these define their own return values. If the dialog closes due to a user-defined mouse up function defined by `DlgMouse()` returning 0 or a negative number, the return value is the negative code or a value greater than any button number if the mouse up function returned 0.

If the user clicks on **OK** or a user-defined button that returns 0, all the variables are updated to their new values. If the user clicks on **Cancel** or a user-defined button that returns a negative value, the variables are not changed.

There is a simple example here.

DlgSlider()

This command adds a slider control that sets an integer value between two user-defined limits. The slider is horizontally orientated for simple use but can be vertically orientated. This command was new in Spike2 version 7.07.

```
Proc DlgSlider(item%, text$|width, lb, rt{, iTick{, flags%{, x{, y}}});
```

item% This sets the item number in the dialog in the range 1 to the number of items.

text\$ A left-justified prompt to display, optionally followed by a vertical bar and tooltip text. If **text\$** is used, the slider will be horizontal and fill the space from the end of the prompt to the right hand side of the dialog.

wide This is an alternative to the prompt. If positive, the slider is horizontal and the value sets the slider width in dialog units; a zero value uses the entire dialog width. A negative value sets the height of a vertical slider and you will need to provide the *x* and *y* values to complete the positioning.

lb, rt Sets the values corresponding to the left/bottom and right/top end of the slider. *lb* can be greater than *rt* but must not be the same.

iTick Optional, with a default value of 0. Enables ticks if not zero, values greater than 0 set the tick spacing and negative values set tick auto-scaling in a 1, 2, 5 sequence.

flags% This sets display options and is the sum of: 1=tooltip value readout during drag, 2=change notifications during drag operation, 4=round all slider values to integers.

x If omitted or zero, the slider is right justified in the dialog. Otherwise positive values set the position of the left side of the slider relative to the left-hand side of the dialog in dialog units and and negative values set the right side position relative to the right-hand side of the dialog.

y If omitted or zero, this takes the value of **item%**. It is the vertical position of the number entry box in dialog units.

The variable passed into `DlgShow()` as argument number **item%** should be a real. The slider starts with the value of the variable if it is in the range. Otherwise, it is limited to the nearer end of the range. Note that the range of values that a slider can return is quantised by the pixel positions that the slider occupies but the returned variable only changes if the slider position is moved or if the original value is out of the slider range.

Tooltips

If you enable the tooltip value readout option in **flags%**, the position of the slider in the control appears in any tip when the mouse is over the slider. If you have set a tip with the **text\$** argument, this tip appears when the mouse is over the prompt. If you add 4 to **flags%**, all displayed positions are integral.

Change function

If you have used `DlgAllow()` to set a change function, you can choose if this is called every time the slider position changes during a drag operation, or only when the drag operation ends.

Keyboard control

The cursor left, right, up and down keys move the slider when it is has the keyboard focus.

DlgString()

This defines a dialog entry that edits a text string. You can limit the characters that you will accept in the string. For simple dialogs, the `wide`, `x` and `y` arguments are not used.

```
Proc DlgString(item%, text$|wide, max%{, legal${, x{,y{,sel$}}});
```

- `item%` This sets the item number in the dialog in the range 1 to the number of items.
- `text$` The prompt to display, optionally followed by a vertical bar and tooltip text.
- `wide` This is an alternative to the prompt. It sets the width in dialog units of the box in which the user types the string. If the width is not given the number entry box has a default width of `max%` or 60, whichever is the smaller.
- `max%` The maximum number of characters allowed in the string.
- `legal$` A list of acceptable characters. See `Input$()` for a full description. If this is omitted, or an empty string, all characters are allowed.
- `x` If omitted or zero, the string entry box is right justified in the dialog. Otherwise positive values set the position of the left side of the box relative to the left-hand side of the dialog in dialog units and negative values set the right side position relative to the right-hand side of the dialog.
- `y` If omitted or zero, this takes the value of `item%`. It is the vertical position of the string entry box in dialog units.
- `sel$` If this string is present, it should hold a list of items separated by vertical bars, for example "one | two | three". The field becomes an editable combo box with the items in the drop down list. This was added at version 5.

The result from this operation is a string of legal characters. The variable passed to `DlgShow()` should be a string. If the initial string set in `DlgShow()` contains illegal characters, they are deleted. If the initial string is too long, it is truncated.

There is a simple example here.

DlgText()

This places non-editable text in the dialog box. This is different from `DlgLabel()` as you do not supply an item number and it does not require a variable in the `DlgShow()` function. It returns an item number (higher than item numbers for matching arguments in `DlgShow()`) that you can use to identify this field in call-back functions, for example `DlgVisible()`. There was no returned value in Spike2 version 4.

```
Func DlgText(text$, x, y{, wide});
```

- `text$` The prompt to display, optionally followed by a vertical bar and tooltip text.
 - `x,y` The position of the bottom left hand corner of the first character in the string, in dialog units. Set `x` to 0 for the default label position (the same as `DlgLabel()`). Otherwise positive `x` values set the position of the left side of the text relative to the left-hand side of the dialog and negative values set the right side position relative to the right-hand side of the dialog.
 - `wide` Normally, the width of the field is set based on `text$`. This optional argument sets the width in dialog units. This allows you to replace the text with a longer string from a call-back function.
- Returns An item number to identify this field for call-back functions.

DlgValue() and DlgValue\$()

These functions can only be used from a dialog call-back function to get and optionally set the value of an item, spinner increment, item prompt or button text. They were new in version 5.

```
Func DlgValue(item%, val);  
Func DlgValue$(item%, val$);
```

item% This identifies the dialog item. For items with arguments in `DlgShow()`, use the `item%` value you set to create the field. For items created with `DlgText()` and `DlgGroup()`, use the returned item number. For buttons use minus the button number. To access the prompt for an item add 1000 to the item number. To access the spinner increment for `DlgReal()` and `DlgInteger()` fields, add 2000 to the item number.

val This optional argument holds the new item value. Use `val` on numeric fields or to set a check box, spinner increment or an item number in a list. Use `val$` to set a prompt, button label or the text of an editable control or to select the first matching item in a list box. It is up to you to make sure the text is acceptable for editable items.

Returns The returned value is the current value of the item before any change. You can use `DlgValue$()` on any item to get the current contents of the field, check box text, button or prompt as a text string. Use `DlgValue()` to collect numeric or check box values.

If there is a problem running the command, for example if the item does not exist, or an argument type is not appropriate for an item, the result is an empty string or the value 0.

You can find an example here.

DlgVisible()

This can only be used from a dialog call-back function to show or hide dialog items. There are two versions of this command. The version with a single argument returns the visible state of an item; the version with two or more arguments sets the visible state of one or more dialog items. When you show or hide an item, any prompt or spin control associated with the item is also shown or hidden. This function was new in version 5.

```
Func DlgVisible(show%, item%|item%[]{|, item%|item%[]...});  
Func DlgVisible(item%);
```

show% Set this to 1 to show the items in the list and to 0 to hide them.

item% An item number of an element of the dialog or an integer array containing a list of item numbers. The item numbers are either the number you set, or the number returned by `DlgText()` or `DlgGroup()`, or `-button`, where `button` is the button number. You cannot access prompts separately from their items as this makes no sense.

Returns When called with a single argument it returns the visible state of the item, otherwise the return value is 0.

DlgXValue()

This creates an editable combo box to collect an x axis value for the current time, result or XY view. The combo box drop-down list is populated with cursor positions and other window values when `DlgShow()` runs. If the current view is not suitable, the list is empty. This control accepts expressions, for example: `(Cursor(1)+Cursor(2))/2`. The matching `DlgShow()` argument is a real number to hold a time in seconds for a time view, or an x axis value for other views. This command was new in version 5.

```
Proc DlgXValue(item%, text$|wide{, x{, y}});
```

item% This sets the item number in the dialog in the range 1 to the number of items.

text\$ The prompt to display, optionally followed by a vertical bar and tooltip text.

wide This is an alternative to the prompt. It sets the width in dialog units of the combo box. If the width is

not given the combo box has a default width of 18 numbers.

- x If omitted or zero, the string entry box is right justified in the dialog. Otherwise positive values set the position of the left side of the box relative to the left-hand side of the dialog in dialog units and and negative values set the right side position relative to the right-hand side of the dialog.
- y If omitted or zero, this takes the value of `item%`. It is the vertical position of the string entry box in dialog units.

Draw()

This optionally positions the current view and allows invalid regions to update. `Draw()` with no arguments on a view that is up-to-date should make no change. The view is not brought to the front. Use `Yield()` to get a text view up to date.

```
Proc Draw({from {, size}});
```

`from` The left hand edge of the window. For a time window, this is in seconds. For a result view, this is in bins. For an XY view, it is in x axis units. For a text view, this sets the top line to display in the view (or as close to the top line as is possible) and scrolls the view horizontally so that the first character position is at the left of the view.

`size` The width of the window in the same units as `from`. A negative `size` is ignored. This argument must be omitted in a text view.

With two arguments, the width is set (unless it is unchanged) and then it is drawn. With one argument, the view scrolls by an integral number of pixels such that `from` is in the first pixel. The following may not move the display if a pixel is more than a second wide:

```
Draw(XLow()+1.0); 'This may scroll by less than 1 second
```

Time views run from time 0 to the maximum time in the view. Result views have a fixed number of bins, set when they are created. XY view axes can be any positive length. The length of a text view is set by lines. You can convert the current position of the selection to lines and columns using `MoveBy()`.

See also:

`DrawAll()`, `XRange()`, `XLow()`, `XHigh()`, `Maxtime()`, `MoveBy()`

DrawAll()

This routine updates all views with invalid regions. Nowadays, calling `Yield()` is a better solution as this also allows the system time to clean up unused resources.

```
Proc DrawAll();
```

See also:

`Draw()`, `Yield()`

DrawMode()

This sets and reads the channel display mode in a time or result view. You can set the display mode for hidden channels. The first command variant returns information, the second is for time views, the third is for result views and the fourth is for sonograms. You can also use `MarkShow()` to set the Marker drawing mode.

```
Func DrawMode(chan%{,mode%});  
Func DrawMode(cSpc, mode%{, dotSz%|binSz%|flags%{, trig%|edge%{, as%}}});  
Func DrawMode(cSpc, mode%{, dotSz% {, opt%|err%{, sort}}});  
Func DrawMode(cSpc, 9{, fftSz%, wnd%, top, range, xInc%, skip%});
```

`chan%` The channel number used to read back information with a negative `mode%`.

`cSpc` A channel specifier or -1 for all, -2 for visible or -3 for selected channels. Setting a draw mode for a bad channel number has no effect.

`mode%` If `mode%` is omitted, the function returns the mode. If negative, see below for return values. Positive values set the display mode. If an inappropriate mode is requested, no change is made. Some modes require additional parameters (for example a bin size). If they are omitted, the last known value is used.

The mode values for setting modes in a time view are:

- 0 The standard drawing mode for the channel. This also sets any marker or marker-derived type channels to display marker code 0 in the standard mode for the channel type.
- 1 Dots mode for events or a waveform. `dotSz%` argument can be used.
- 2 Lines mode. Shows event data as vertical ticks on a horizontal line. If `dotSz%` is present and zero, the horizontal line is not drawn.
- 3 Waveform mode. Draws straight lines between waveform and WaveMark points. No WaveMark codes are displayed (faster than WaveMark mode).
- 4 WaveMark mode. Only use this if you wish to see WaveMark codes as it takes longer to draw than Waveform mode.
- 5 Rate mode. `binSz` sets the width of each bin in seconds.
- 6 Mean frequency mode. `binSz` sets the time period.
- 7 Instantaneous frequency mode. `dotSz%` and `as%` can be used.
- 8 Raster mode. `trig%` sets the trigger channel, `dotSz%` is used.
- 9 Sonogram mode. In this mode the `fftSz%`, `wnd%`, `top`, `range`, `xInc%` and `skip%` arguments are allowed (or can be omitted from the right).
 - `fftSz%` The block size for the Fourier Transform used to generate the sonogram. Allowed values are 16, 32, 64, 256, 512, 1024, 2048 and 4096. Intermediate values set the next lower allowed value.
 - `wnd%` The window applied to the data. 0 = no window, 1 = Hanning, 2 = Hamming. Values 3 to 9 set Kaiser windows with -30 dB to -90 dB sideband ripple in steps of 10 dB.
 - `top` The signal dB value relative to 1 bit at the ADC input to show as the maximum density output. Signal values above `top` dB are shown in maximum intensity. Values 0.0 to 100.0 are allowed.
 - `range` The range in dB of output data below `top` that is mapped into the grey scale. Output below (`top-range`) is shown as minimum intensity. Values greater than 0.0 and up to 100.0 are allowed.
 - `xInc%` The number of pixels (1 to 100) to step across the screen before calculating the next set of sonogram values. This is normally 1.
 - `skip%` Normally 0. Set 1 to use one data block for each vertical strip of sonogram to reduce the calculation time for large files (but only shows samples of the sonogram). Values 0 and 1 are allowed.
- 10 WaveMark overdraw mode.
- 11 Same as mode 6, but display rate per minute rather than per second.
- 12 Same as mode 7, but display rate per minute rather than per second.
- 13 Cubic spline mode for waveform, WaveMark and RealMark channels.
- 14 Text mode for TextMark channels. `dotSz%` is used.
- 15 State mode for marker channels. `flags%` is used.
- 16 Skyline mode for waveform, RealWave and RealMark channels.

For a result view channel the modes are listed below.

- 0 The standard drawing mode for the result view.
- 1 Draw as a histogram. `err%` can be used.
- 2 Draw as a line. `err%` can be used.
- 3 Draw as dots. `err%` and `dotSz%` can be used.
- 4 Draw as a skyline. `err%` can be used.
- 8 Draw raster as lines, `dotSz%` sets the line length, `opt%` is used.
- 9 Draw raster as dots, `dotSz%` sets the dot size, `opt%` is used.

13 Cubic spline mode. `err%` can be used.

- `dotSz%` Sets the dot or tick size to use in units of the the pen width set for data in the Edit menu Preferences Display tab. 0 is the smallest size available. The maximum size is 10. Use -1 for no size change.
- `binSz` Sets rate histogram bin width and the mean frequency smoothing period.
- `flags%` For markers in State mode, this sets the additional information to display. 0=no extra information. Add 1 to draw the state number, 2 to display TextMark text.
- `trig%` This is the trigger channel for time view raster displays. We assume that you do not want to display a level event channel in raster mode.
- `edge%` Sets the edges of level event data to use for mean and instantaneous frequency and rate modes. 0 = both edges (default), 1 = rising edges and 2 = falling edges. This is ignored (and should be set to 0) for non-level event channels.
- `as%` Used with instantaneous frequency mode and for RealMark data drawn as a waveform to determine how the data is drawn and measured. 0=Default, 1=Dots, 2=Line, 3=Skyline.
- `opt%` Used in result view raster channels. The default is 0. Add 1 for horizontal line in raster line mode. Add 2 for y axis as time of sweep or variable value in raster modes. Add 16 to show symbols. For backwards compatibility, if `sort%` is omitted, add 4 to select sort value 1 and 8 to select sort value 2. These values (4 and 8) are not returned when `mode%` is -12, use -15 to get the sort value.
- `err%` Result view error drawing style: 0=none (default), 1=1 SEM, 2=2 SEM, 3=SD.
- `sort%` Result view raster sort mode. 0=time (default), 1-4 = use sort variables 1 to 4.

Returns If a single channel is set it returns the previous `mode%`. For multiple channels or an invalid call, it returns -1. Negative `mode%` values return drawing parameters:

```
-1 Reserved  -4 trig%  -7 wnd%  -10 xInc%  -13 err%  -16 as%
-2 dotSz%   -5 edge%  -8 top   -11 skip%  -14 sort%
-3 binSz    -6 fftSz% -9 range -12 opt%  -15 flags%
```

See also:

Channel specifiers, `ChanIndex()`, `Draw()`, `MarkShow()`, `SetEvtCrl()`, `SetPSTH()`, `ViewStandard()`, `XYDrawMode()`

Dup()

This gets the view handle of a duplicate of the current view or the number of duplicates. Duplicated views are numbered from 1 (1 is the original). If a duplicate is deleted, higher numbered duplicates are renumbered. See `WindowDuplicate()` for more information.

```
Func Dup( {num%} );
```

`num%` The number of the duplicate view to find, starting at 1. You can also pass 0 (or omit `num%`) as an argument, to return the number of duplicates.

Returns If `num%` is greater than 0, this returns the view handle of the duplicate, or 0 if the duplicate does not exist. If `num%` is 0 or omitted, this returns the number of duplicates. The following code illustrates the use of `Dup()`.

```
var maxDup%, i%, dvh%;      'declare variables
maxDup% := Dup(0);         'Get maximum numbered duplicate
for i% := 1 to maxDup% do  'loop round all possible duplicates
    dvh% := Dup(i%);       'get handle of this duplicate
    if (dvh% > 0) then     'does this duplicate exist?
        PrintLog(view(dvh%).WindowTitle$()+"\n"); 'print window title
    endif;
next;
```

See also:

`App()`, `View()`, `WindowDuplicate()`

DupChan()

This returns the number of duplicates of a channel in the current time or result view, or the channel number of the n^{th} duplicate, or it identifies the channel on which a duplicate channel is based, or it identifies which duplicate of an original channel this is.

```
Func DupChan(chan%{, num%});
```

chan% A channel number in the current view. The channel need not exist. If this channel is a duplicate, the command behaves as though you passed the original channel on which the duplicates are based.

num% If greater than 0, this is the index of a duplicate channel in the range 1 to the number of duplicates. Use 0 to return the original channel that was duplicated. Use -1 as an argument (or omit **num%**), to return the number of duplicates of the channel. Use -2 as an argument to return which duplicate a channel is: 0 means not a duplicate, 1 upwards is the duplicate index.

Returns If **num%** is greater than 0, this returns the channel number of the duplicate, or 0 if the duplicate does not exist. If **num%** is 0, this returns the channel number that **chan%** was duplicated from or **chan%** if it is the original or **chan%** does not exist. If **num%** is -1 or omitted, this returns the number of duplicates. If **num%** is -2, this returns the duplicate index of **chan%** (0 if not a duplicate, 1 for duplicate a and so on).

See also:

ChanDuplicate()

E

EditClear()

In a result view, this command zeros the histogram data and removes any raster information. In a text view, this command deletes any selected text.

```
Func EditClear();
```

Returns The function returns 0 if nothing was deleted, otherwise it returns the number of items deleted or 1 if the number is not known, or a negative error code.

See also:

EditCut(), EditFind(), EditPaste()

EditCopy()

This command copies data from the current view to the clipboard or copies a string to the clipboard. Time views copy as a bitmap, as a scalable image and as text in the format set by `ExportRectFormat()`, `ExportTextFormat()`, `ExportChanFormat()` and `ExportChanList()`. Result and XY views can copy as a bitmap, text and as a scalable image. Multimedia views copy as a bitmap. Text windows copy as text.

```
Func EditCopy({as%|text$});
```

as% Sets how to copy data when several formats are possible. If omitted, all formats are used. It is the sum of: 1=Copy as a bitmap, 2=Copy as a scalable image (Windows metafile), 4=Copy as text

text\$ A text string to place on the clipboard. This can be useful when you want to move results to a different program.

Returns It returns 0 if nothing was placed on the clipboard, or the sum of the format specifiers for each format used.

See also:

EditSelectAll(), ExportTextFormat(), ExportChanFormat(), ExportRectFormat(), ExportChanList(), EditCut(), EditPaste()

EditCut()

This command cuts data from the current view to the clipboard and deletes the original.

```
Func EditCut({as%});
```

as% This optional argument sets how data is cut and copied to the clipboard. Currently only text may be cut. If omitted, all allowed formats are used.

- 1 Cut and copy as a bitmap, has no effect
- 2 Cut and copy as a scalable image (metafile), has no effect
- 4 Cut text to clipboard

Returns 0 if nothing was placed on the clipboard, or the sum of the format specifiers for each format that was used. The only possible values now are 0 and 4.

See also:

EditCopy(), EditClear()

EditFind()

In a text view, this command searches from the selection point for the next occurrence of the specified text and selects it if found. This is the same as the Find Text dialog.

```
Func EditFind(find${, dir%{, flags%}});
```

find\$ The text to search for. May include regular expressions if 4 is added to **flags%**.

dir% Optional. 0=search backwards, 1=search forwards (default), 2=wrap around.

flags% Optional, taken as 0 if omitted. The sum of: 1=case sensitive, 2=find complete words only, 4=regular expression search (**dir%** may not be 0).

Returns 1 if found, 0 if not found.

See also:

Find text dialog, EditReplace()

EditPaste()

Any text clipboard contents are either inserted into the text at the selection, replacing any selected text or are returned in a text variable. You may only paste into a text window when the clipboard contains text.

```
Func EditPaste({text$});
```

text\$ If present, any text in the clipboard is returned in this string. If absent, clipboard text is pasted to the current text view.

Returns If **text\$** is present the return value is 1 if the clipboard holds text, 0 if it does not. If **text\$** is absent it returns the number of characters pasted into the view.

See also:

EditCopy(), EditClear(), EditCut()

EditReplace()

In a text view, this command checks if the selection matches a pattern and replaces it if it does, then it searches for the pattern again. This is the same as the Replace Text dialog.

```
Func EditReplace(find${, repl${, dir%{, flags%}}});
```

find\$ The text to search for. May include regular expressions if 4 is added to **flags%**.

`repl$` Optional replacement text, taken as an empty string if omitted.
`dir%` Optional. 0=search backwards, 1=search forwards (default), 2=wrap around.
`flags%` Optional, taken as 0 if omitted. The sum of: 1=case sensitive, 2=find complete words only, 4=regular expression search (`dir%` may not be 0).

Returns The sum of: 1 if found a new match, 2 if replaced the original selection.

See also:

Replace text dialog, `EditFind()`

EditSelectAll()

This function selects all items in the current view that can be copied to the clipboard. This is the same as the Edit menu Select All option.

```
Func EditSelectAll();
```

Returns It returns the number of selected items that could be copied to the clipboard.

See also:

`EditCopy()`, `EditClear()`, `EditCut()`, `MoveBy()`, `MoveTo()`

Error\$()

This function converts a negative error code returned by a function into a text string.

```
Func Error$(code%);
```

`code%` A negative error code returned from a Spike2 function.

Returns It returns a string that describes the error.

See also:

`Debug()`, `Eval()`, `Print$()`, `PrintLog()`

Eval()

This evaluates the argument and converts the result into text. The text is displayed in the Script window or Evaluate window message area, as appropriate, when the script ends. This overrides any subsequent script runtime error message.

```
Proc Eval(arg);
```

`arg` A real or integer number or a string.

If you use `Eval()` it will suppress any run-time error messages as it uses the same mechanism as the error system. A common use of `Eval()` in a script is to report an error condition during debugging, for example:

```
if val<0 then Eval("Negative value"); Halt; endif;
```

Another use of `Eval()` is in the Script menu Evaluate window to see the result returned by a function or expression, as in these examples:

```
Eval(FileDelete(myfile$)); ' display 1 or a negative error code  
Eval(Error$(-1531));      ' give string for error code if known
```

See also:

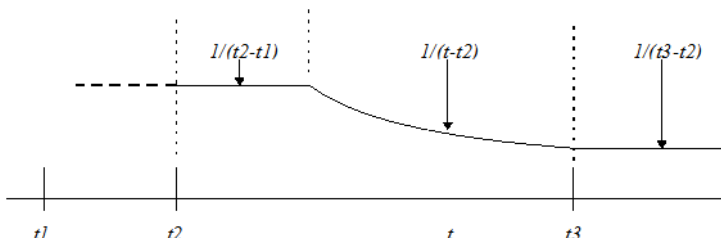
`Debug()`, `Error$()`, `Print()`, `PrintLog()`

EventToWaveform()

This function creates a waveform channel from an event channel. The waveform channel contents depend on the frequency of the events. You can convert based on instantaneous or smoothed frequency. `VirtualChan()` also converts events to a waveform.

Instantaneous frequency

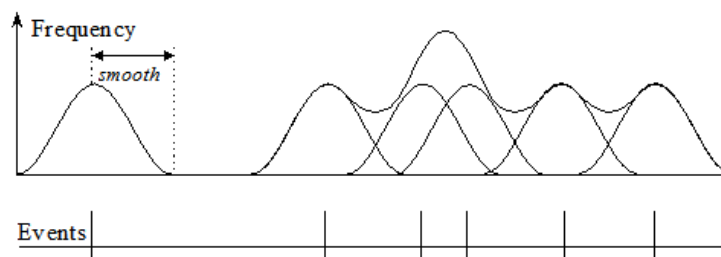
Spike2 calculates instantaneous frequency as the reciprocal of event intervals. Between events, the interval from the last event to the current time is compared with the interval between the last two events. If it is less, the reciprocal of the last interval is used. If it is more, the reciprocal of the interval from the last event is used.



The diagram shows the result for three events at times t_1 , t_2 and t_3 . From t_1 to t_2 , the result depends on events before t_1 . From t_2 to $t_2+(t_2-t_1)$, the result is set by the previous interval. From this point onwards to t_3 , the output reduces until at t_3 it becomes $1/(t_3-t_2)$.

Smoothed frequency

Spike2 calculates smoothed frequency by replacing (convoluting) each event with a waveform of unit area. The built-in waveforms are symmetrical about the event time. If the event is at time t , the waveforms extend from time $t-\text{smooth}$ to $t+\text{smooth}$. You can also provide your own waveforms, and these need not be symmetric.



The diagram shows the result of using the built-in raised cosine waveform. The area under the curve for each spike is 1. You would normally use a smoothing period that covered several events to obtain a smooth output.

The command

```
Func EventToWaveform(smooth, eChan%, wChan%, wInt, sTime, eTime,
    maxF, meanF {,query% {,wave[]|wave%{, nPre}}});
```

- `smooth` The smoothing period in seconds. Set this 0 or negative for instantaneous frequency, otherwise an event at time t is spread over a time range $t-\text{smooth}$ to $t+\text{smooth}$ seconds for symmetric functions. If you omit the `wave` argument, the smoothing function is a raised cosine bell.
- `eChan%` The channel number to use as a source of event times. If the channel is a marker, and a marker filter is set, only filtered events are visible.
- `wChan%` The channel number to create as a waveform in the range 1 to the maximum allowed channels in the file.
- `wInt` The desired sampling interval for the new channel in seconds. Values outside the range 0.000001 to 1000 are errors. This interval is rounded to the nearest multiple of the microseconds per time unit set for the file. Use `BinSize()` to return the actual sampling interval. Before version 4.03, `wInt` was rounded to the nearest multiple of the base ADC convert interval for the file. Older Spike2 versions will not open the file if you set an interval that they cannot achieve.
- `sTime` The start time for output of the waveform data.
- `eTime` The last time for waveform output will be less than or equal to this time. The command processes events from $sTime-\text{smooth}$ to $eTime+\text{smooth}$.
- `maxF` The output is stored as if it were a sampled signal. That is as a 16-bit integer in the range -32768 to 32767. `maxF` sets the frequency that corresponds to the 32767 value. It is the maximum frequency that can be represented in the result. Frequencies higher than this are limited to `maxF`.

meanF This sets the frequency that corresponds to 0 in the 16-bit waveform data. Most users set this to 0. The lowest frequency that can appear in the result is $\text{maxF} - 2 * (\text{maxF} - \text{meanF})$. Frequencies lower than this are limited to this value.

If the result lies in the range $\text{freq} \pm \text{delta}$, you get the best resolution by setting **maxF** to $\text{freq} + \text{delta}$ and **meanF** to freq .

query% Set this to 0 or omit it to query overwriting an existing channel. If this is present and non-zero, an existing channel is overwritten with no comment.

wave% This optional argument is used when **smooth** is greater than 0 and sets the smoothing function: 0=rectangular, 1=triangular, 2=raised cosine, 3=Gaussian (to 4 sigmas). If omitted, raised cosine smoothing is used. These functions are the same as are used for virtual channels.

wave[] If you provide an array, it is scaled to span **smooth** seconds. The sum of the waveform values must be in the range $1.2\text{e-}38$ to $3.4\text{e}+38$. If **nPre** is omitted or negative, the smoothing function is symmetrical and you supply the right-hand side only. You can set any array size; the built-in functions use 1000 points. This example matches the raised cosine generated with **wave%** set to 2.

```
var wave[1000], i%, nPre := -1.0;
const pi := 3.14159;
for i% := 0 to 999 do wave[i%]:=1 + Cos(i%*pi/1000) next;
```

nPre This sets the index in **wave[]** that matches the event time for asymmetric smoothing functions. The entire **wave[]** array is **smooth** seconds wide. If **nPre** is omitted or negative, a symmetric function is assumed. This example generates a typical asymmetric smoothing function in the **wv[]** array:

```
var wv[400], i%, nPre;
const a:=40, b:=80; 'The time constants in points
for i%:=0 to 399 do wv[i%]:=exp(-i%/a)*(1-exp(-i%/b)) next;
nPre := b*Ln(1+a/b); 'Calculate position of maximum
```

Returns The function returns 0 unless the user decided not to overwrite a channel, in which case the result is a negative error code. You can also get error -5799 if you run out of memory (unlikely) or disk space.

Uses of EventToWaveform

This can be used to investigate frequency changes in an event train. If you have an event channel of heart beats at 70 beats per minute (1.17 Hz) and you are seeking a respiration-induced modulation at about 0.25 Hz, you can convert the heartbeats into a waveform with this command. The result would be a constant level of 1.17 Hz, plus a small oscillation around 0.25 Hz. The oscillation can be detected with **SetPower()**.

For 0.01 Hz resolution with **SetPower()** you need 100 seconds of data per power spectrum block. With a 1024 point transform size, a **wInt** of 0.1 Hz gives a block size of 102.4 seconds. The **SetPower()** result will have 512 bins, spanning 0 to 5.00 Hz in steps of 0.01 Hz. A **smooth** period of 5-10 seconds would be appropriate.

See also:

Virtual channel smoothing, **BinSize()**, **SetPower()**, **VirtualChan()**

Exp()

This function calculates the exponential function for a single value, or replaces a real array by its exponential. If a value is too large, overflow will occur, causing the script to stop for single values, and a negative error code for arrays.

```
Func Exp(x|x[]{|[]...});
```

x The argument for the exponential function or an array of real values.

Returns With an array, the function returns 0 if all was well, or a negative error code if a problem was found (such as overflow). With an expression, it returns the exponential of the number.

See also:

Abs(), **ATan()**, **Cos()**, **Frac()**, **Ln()**, **Log()**, **Max()**, **Min()**, **Pow()**, **Rand()**, **Round()**, **Sin()**, **Sqrt()**, **Tan()**, **Trunc()**

ExportChanFormat()

This sets the channel text format used by `ExportTextFormat()`, `FileSaveAs()` and `EditCopy()`. It is equivalent to the Channel type parameters section of the File Dump Configuration dialog. See `ExportTextFormat()` for the command to reset the format.

```
Func ExportChanFormat(type%, synop%, data%{, as%});
```

`type%` The channel type to set the format for. Types 2, 3 and 4 share the same settings. Types 1 and 9 also share the same settings.

1 Waveform	4 Level (Evt+-)	7 RealMark
2 Event (Evt-)	5 Marker	8 TextMark
3 Event (Evt+)	6 WaveMark	9 RealWave

`synop%` Set this non-zero to enable synopsis output for this channel type.

`data%` Set this non-zero to enable data output for this channel type.

`as%` This sets the format for markers and descendent types. It is ignored and can be omitted for waveform and event channels. If omitted or set to 0, the channels are treated as their own type. The codes are the same as for the `type%` field, but the only useful values are 2,3 or 4 to output as if the channel were an event, or 5 to dump the channel as if it were a marker. All other values are ignored.

Returns 0 if all was OK or a negative error code.

See also:

`FileSaveAs()`, `ExportChanList()`, `ExportTextFormat()`

ExportChanList()

This command sets a time range and a channel list to export for use by `FileSaveAs()` and `EditCopy()`. The current view must be a time view. There are two command variants. The first with one or no arguments clears the channel list. The second, with a channel specification, adds a set of channels and a time range to a list., suitable for use with `ExportTextFormat()`. When used with `ExportRectFormat()` or with an external exporter, only the first set of channels and time range is used. To write different segments of a file to separate output files, clear the list with the first command variant before setting each segment with the seconds variant, then output the data with `FileSaveAs()` or `EditCopy()`. See `ExportTextFormat()` and `ExportRectFormat()` for examples.

```
Proc ExportChanList({flags%});  
Proc ExportChanList(sTime, eTime, cSpc {,cSpc...});
```

`sTime` The start time of the range of data to save.

`eTime` The end time of the range of data to save.

`cSpc` A channel specifier for the channels to export.

`flags%` This affects data written by `FileSaveAs()` with `type%` of 0 and is the sum of:

- 1 Time shift data so that the earliest `sTime` appears at time 0 in the output file.
- 2 Write RealWave channels as waveform data for backwards compatibility.

WARNING: Remember to clear the list before using the second command variant unless you really want to add a new time range and channel specification to the end of the existing list.

See also:

Channel specifiers, `FileSaveAs()`, `ExportChanFormat()`, `ExportTextFormat()`, `ExportRectFormat()`, `EditCopy()`

ExportRectFormat()

This command sets the text export format for use by `FileSaveAs()` and `EditCopy()` in a time view. This is an alternative to using `ExportTextFormat()` and generates equally spaced output in time for all channels. If used with no parameters, this will reset the rectangular text output configuration to the default of 100 Hz output, linear interpolation, double-quotes delimiter and tab for the separator. Otherwise at least 2 arguments must be supplied.

```
Proc ExportRectFormat({freq, inter%{, delim${, sep$}}})
```

`freq` This sets the number of output points per channel generated per second.

`inter%` This sets the interpolation method:

- 0 Use the data point nearest to the time.
- 1 Linear interpolation between the points either side.
- 2 Cubic spline interpolation (waveform data only).
- 3 Cubic spline interpolation if drawn splined, else linear (waveforms only).

`delim$` This sets the character used to delimit the start and end of text strings. If omitted, text strings are surrounded by quote marks, for example, "Volts".

`sep$` The character to separate multiple data items on a line. If omitted, a Tab is used.

The following example exports channels 1, 2 and 3 at 200 Hz from times 10 seconds to 30 seconds as a text file. The times are shifted so that the first data item is at time 0. Waveform channels are mapped to 200 Hz using cubic spline interpolation.

```
ExportChanList(1);           'Clear export list, sets zero shift
ExportChanList(10,30,1,2,3); 'Choose channels 1, 2 and 3
ExportRectFormat(200, 2);    '200 Hz, cubic spline format
FileSaveAs("rect.txt", 1);   'Export as text file, and...
EditCopy();                  '...also copy to the clipboard
```

See also:

`FileSaveAs()`, `ExportChanList()`, `ExportTextFormat()`, `EditCopy()`,

ExportTextFormat()

This command sets the text export format for use by `FileSaveAs()` and `EditCopy()` in a time view. It is equivalent to the top section of the File Dump Configuration dialog. The form with no arguments resets the dialog to enable all check boxes, sets the columns to 1, sets the string delimiter to a double quote mark, sets the separator to a tab character, and sets all channel types to be treated as themselves.

```
Proc ExportTextFormat(head%, sum%, cols%{, delim${, sep$}});  
Proc ExportTextFormat();
```

`head%` If this is non-zero, Spike2 will output the header.

`sum%` If this is non-zero, Spike2 outputs the channel summary information.

`cols%` The number of columns to use when writing waveforms and event times.

`delim$` This sets the character used to delimit the start and end of text strings. If omitted, text strings are surrounded by quote marks, for example, "Volts".

`sep$` The character to separate multiple data items on a line. If omitted, a Tab is used.

The following example exports channels 1 and 2 from 90.0 to 100.0 seconds and from 110 to 120 seconds as a file 30 seconds long called `fred.smr`.

```
ExportChanList(1);           'Clear export list, sets zero shift
ExportChanList(90, 100, 1, 2); 'set channels 1 and 2
ExportChanList(110,120, 1, 2); 'add 10 more seconds after 10s gap
ExportTextFormat();          'reset the file dump dialog
ExportChanFormat(1,0,1);     'turn off synopsis for waveform data
ExportTextFormat(0, 0, 1);    'No header or summary
FileSaveAs("fred.smr", 0);    'export to new data file
```

See also:

EditCopy(), ExportChanList(), ExportChanFormat(), ExportRectFormat(), FileSaveAs()

F**File...()****FileApplyResource()**

This applies a resource file to the current time, result or XY view. If a time view is duplicated, all other duplicates are deleted, then the resource file is applied, which may create duplicates. The current view handle is not changed. Handles of duplicate views are not preserved, even if the resource file creates the same number of duplicates.

```
Func FileApplyResource(name$);
```

name\$ The resource file to apply. You must include the file extension and required path. Remember that \ in a string must be entered as \\ or use /. If the name is "" or contains a "*" or a "?", the user is prompted for a file.

Returns The number of modified views (usually 1 unless the resource file generates duplicates), 0 if the resource file did not contain suitable information, -1 if the user cancelled the file dialog, -2 if the file could not be found or -3 if there is any other problem.

See also:

FileGlobalResource(), FileOpen(), FileSave(), FileSaveResource()

FileClose()

This is used to close the current window or external file. You can supply an argument to close all windows associated with the current time or result view or to close all the windows belonging to the application.

```
Func FileClose({all% {,query%}});
```

all% This argument determines the scope of the file closing. Possible values are:

- 1 Close all windows except loaded scripts and debug windows.
- 0 Close the current view. This is the same as omitting **all%**.
- 1 Close all windows associated with the current view.

query% This determines what happens if a window holds unsaved data:

- 1 Don't save the data or query the user
- 0 Query the user about each window that needs saving. If the user chooses Cancel, the operation stops, leaving all unclosed windows behind. This is the same as omitting **query%**.

Returns The number of views that have not been closed. This can occur if a view needs saving and the user requests Cancel, or if the view was handling a user-defined mouse function.

If the current view is a time view and you have been sampling data, you must call `SampleStop()` before using this command to set a file name. If you do not, this is equivalent to aborting sampling and your data file will not be saved. Use `FileSaveAs()` just before calling `FileClose()` to set a file name.

Do not use `View(fh%).FileClose()` if **fh%** is the current view; you will get an error when Spike2 tries to restore the current view. Use `View(fh%);FileClose()` instead.

See also:

FileOpen(), FileSave(), FileSaveAs(), FileNew(), SampleStop()

FileComment\$()

This function accesses the file comments in the file associated with the current time view. Files have five comment strings. Each string is up to 79 characters in length.

```
Func FileComment$(n% {,new$});
```

n% The number of the file comment line in the range 1 to 5

new\$ If present, the command replaces the existing comment with *new\$*.

Returns The comment at the time of the call or a blank string if *n%* is out of range.

See also:

ChanComment\$()

FileConvert\$()

This function converts a data file from a “foreign” format into a Spike2 data file. The range of foreign formats supported depends on the number of import filters in the `import` folder (inside the folder where Spike2 was installed). The convert process follows the description for the interactive File menu Import command. However, there is no progress dialog, so you will have to wait patiently for the result.

```
Func FileConvert$(src${, dest${, flag%{, &err%{, cmd$}}});
```

src\$ This is the name of the file to convert. The file extension is used to determine the file type (unless *flag%* bit 0 is set). Known file extensions and configuration information can be found [here](#).

dest\$ If present, it sets the destination file. If this is not a full path name, the name is relative to the current directory. If you do not supply a file extension, Spike2 appends ".smrx". If you set any other file extension, Spike2 cannot open the file as a data file. If you omit this argument, the converted file is written to the same name and path as the source file with the file extension changed to .smrx.

flag% This argument is the sum of the flag values: 1=Ignore the file extension of the source file and try all possible file converters, 2=Allow user interaction if required (otherwise sensible, non-destructive defaults are used for all decisions).

err% Optional integer variable that is set to 0 if the file was converted, otherwise it is returned holding a negative error code.

cmd\$ Optional string holding parameters that control the importer. The string is of the form: `name1=value1;name2=value2;name3=value3` where the names are case insensitive. We append `;AppVer=Spike2,800` (the 800 stands for the version of Spike2 multiplied by 100) to the end of the string. It is entirely up to the importers what names they recognise. For more information about specific importers [click here](#) and then find the importer in the table. If the importer supports a command line, the **cmd\$** column contains Yes. Click the Yes for details of supported keywords.

Before the string is passed to an importer it is scanned for parameters that apply generally. At the moment, one one such option exists:

Name	Description
------	-------------

dll	You can use this option to select a specific import DLL. If you do not supply this, Spike2 will use the first importer it finds that supports a file extension that matches the extension of the file set by <i>src\$</i> . Most data formats have a unique extension, but some have a wide range of extensions. For example, to force the use of the CED binary importer you would set <code>cmd\$:= "dll=binary"</code> as the argument. You can find the DLL name from the table of importers for the File menu Import command, or record your actions.
-----	---

Returns The full path name of the new file or an empty string if there was no output.

See also:

FileOpen(), FilePath\$, FilePathSet(), FileList(), Import command

FileCopy()

This function copies a source file to a destination file. File names can be specified in full or relative to the current directory. Wildcards cannot be used.

```
Func FileCopy(src$, dest${, over%});
```

src\$ The source file to copy to the destination. This file is not changed.

dest\$ The destination file. If this file exists you must set **over%** to overwrite it.

over% If this optional argument is 0 or omitted, the copy will not overwrite an existing destination file. Set to 1 to overwrite.

Returns The routine returns 1 if the file was copied, 0 if it was not. Reasons for failure include: no source file, no destination path, insufficient disk space, destination exists and insufficient rights.

See also:

BRead(), BWrite(), FileDelete(), FileOpen(), ProgRun()

FileDate\$()

This function returns a string holding the date when the data file was sampled with an optional time offset. Use `FileTimeDate()` to get the date as numbers. The current view must be a time view. The first 5 arguments are exactly the same as for the `Date$()` command. The date was not stored in the data file before version 4.03. If there is no date, the result is an empty string.

```
Func FileDate$({dayF%{, monF%{, yearF%{, order%{, sep${, offs%}}}}});
```

dayF% See the `Date$()` command for the first 5 arguments.

offs% This is a time offset from the start of the file, in seconds. It is treated as zero if omitted or if it is negative. You might use the offset if you wanted to write out the date that something happened in a long term recording.

Returns The date as a string.

See also:

Date\$(), FileTimeDate(), FileTime\$()

FileDelete()

This deletes one or more files. File names can be specified in full or relative to the current directory. Windows file names look like `x:\folder1\folder2\foldern\file.ext` or `\\machine\folder1\folder2\foldern\file.ext` across a network. If a name does not start with a `\` or with `x:\` (where `x` is a drive letter), the path is relative to the current directory. Beware that `\` must be written `\\` in a string passed to the compiler.

```
Func FileDelete(name$[]|name${, opt%});
```

name\$ This is either a string variable or an array of strings that holds the names of the files to delete. Only one name per string and no wildcard characters are allowed. If the names do not include a path they refer to files in the current directory.

opt% If this is present and non-zero, the user is asked before each file in the list is deleted. You cannot delete protected or hidden or system files.

Returns The number of files deleted or a negative error code.

See also:

FilePath\$(), FilePathSet(), FileList()

FileGlobalResource()

This function is equivalent to the Global Resources dialog. Please see the documentation for the dialog for a full explanation. The function has two forms: the first is for setting values, the second is to read them back:

```
Func FileGlobalResource(flags%, loc%, name${, path$});  
Func FileGlobalResource(&loc%, &name$, &path$);
```

`flags%` This is the sum of the following values:

- 1 Enable the use of global resources. If unset, no global resources are used.
- 2 Only use global resources if a data file has no associated resource file.
- 4 Only use if the data file is within the path set by `path$`.

`loc%` The location of the global resource file. 0=the folder that Spike2 was run from, 1=search the data file folder, then the Spike2 folder, 2=the data file folder only.

`name$` The name of the global resource file excluding the path and file extension.

`path$` The file path within which to use the global resources. If you omit this argument when setting values, the current path does not change.

Returns When setting values, the return value is 0 or a negative error code. When reading back values, the return value is the `flags%` argument.

See also:

Global Resources dialog, `FileApplyResource()`, `FileOpen()`, `FileSave()`, `FileSaveResource()`

FileList()

This function gets lists of files and sub-directories (folders) in the current directory and can also return the path to the parent directory of the current directory. This function can be used to process all files of a particular type in a particular directory.

```
Func FileList(names$[]|&name$, type%{, mask$});
```

`name$` This is either a string variable or an array of strings that is returned holding the name(s) of files or directories. Only one name is returned per string.

`type%` The objects in the current directory to list. The Parent directory is returned with the full path, the others return the name in the current directory. Values are:

- | | |
|-------------------------------------|--|
| -3 Parent directory | 2 Output sequence files (*.pls) |
| -2 Sub-directories | 3 Spike2 script files (*.s2s) |
| -1 All files | 4 Spike2 result view files (*.srf) |
| 0 Spike2 data files (*.smr, *.smrx) | 6 Spike2 configuration files (*.s2c, *.s2cx) |
| 1 Text files (*.txt) | 12 XY view data file (*.sxy) |

`mask$` This optional string limits the names returned to those that match it; * and ? in the mask are wildcards. ? matches any character and * matches any 0 or more characters. Matching is case insensitive and from left to right.

Returns The number of names that met the specification or a negative error code. This can be used to set the size of the string array required to hold all the results.

Version 8 changes

If you select data or configuration files using the `type%` argument, you will get both old and new format files. This is done so that old scripts written for earlier versions of Spike2 and that use `type%` to specify data or configuration files will still work. You can use the `mask$` argument to be more specific. For example, if you only wanted old-format data files you would set `mask$` to "*.smr".

If your script needs to specify part of the file name, but needs to list both old and new files you could specify the mask as: "may*.smr?" and this would list both may123.smr and may124.smr, however you would

also list `may999.smrz`.

See also:

`FilePath$()`, `FilePathSet()`, `FileDelete()`, `FileName$()`

FileName\$()

This returns the name of the data file associated with the current view (if any). You can recall the entire file name, or any part of it. If there is no file the result is an empty string.

Func `FileName$({mode%});`

`mode%` If present, determines what to return, if omitted taken as 0.

- 0 Or omitted, returns the full file name including the path
- 1 The disk drive/volume name
- 2 The path section, excluding the volume/drive and the name of the file
- 3 The file name up to but not including the last . excluding any trailing number.
- 4 Any trailing numbers from 3.
- 5 The end of the file name from the last dot.

Returns A string holding the requested name, or a blank string if there is no file.

See also:

`FileList()`, `FilePath$()`, `FilePathSet()`, `SampleConfig$()`, `SampleSequencer$()`

FileNew()

This is equivalent to the File menu New command. It creates a new window and returns the handle. You can create visible or invisible windows. Creating an invisible window lets you set the window position and properties before you draw it. The new window is the current view and if visible, the front view. Use `FileSaveAs()` to name created files.

Func `FileNew(type%{, mode%{, upt, tpa%, maxT{, nChan%{, big%}}});`

`type%` The type of file to create:

- 0 A Spike2 data file based on the sampling configuration, ready for sampling. This may open several windows, including the floating command window and the sequencer control panel. Use `SampleStart()` to begin sampling and `SampleStop()` to stop sampling before calling `FileSaveAs()` to give the new file a name. During sampling, type 0 data files are saved in the folder set by the Edit menu preferences or the `FilePathSet()` command. Use `FileSaveAs()` command after `SampleClose()` to move the data file to its final position on disk.
- 1 A text file in a window
- 2 An output sequence file in a window
- 3 A Spike2 script file
- 7 An empty Spike2 data file (not for sampling). `upt%`, `tpa%` and `maxT` must be supplied. You can use `ChanNew()`, `ChanWriteWave()`, `ChanSave()`, `MemSave()` or to add data. Use `FileSaveAs(name$, -1)` to name the new file.
- 12 An XY view with one (empty) data channel. Use `XYAddData()` to add more data and `XYSetChan()` to create new channels.

`mode%` This optional argument determines how the new window is opened. The value is the sum of these flags. If the argument is omitted, its value is 0. The flags are:

- 1 Make the new window(s) visible immediately. If this flag is not set the window is created, but is invisible.
- 2 For data files, if the sampling configuration holds information for creating additional windows or channels, use it. If this flag is not set, data files extract enough information from the sampling configuration to set the sampling parameters for the data channels. You can use the `ViewLink()` command to identify additional views or channels.
- 4 Show the spike shape setup dialog if there are WaveMark channels in the sampling configuration for a data file. If the spike shape dialog appears, this function does not return until the user closes

the dialog.

- upt** Only for `type% = 7`. The microseconds per unit time for the new file. This allows values in the range 0.001 to 32767. If `upt` is non-integral values, Spike2 versions prior to 4.02 will not open the file. This sets the time resolution of the new file and the maximum length of the file, in time. There are a maximum of $2^{31}-1$ time units in a file, so a 1 us resolution limits a file to 30 minutes in length.
- tpa%** Only for `type% = 7`. The time units per ADC conversion for the new file. You can set values in the range 1 to 32767. The available sampling intervals for waveform data in the new file are $n * \text{upt}\% * \text{tpa}\%$ microseconds where n is an integer. This value should be set to 1 in all modern systems.
- maxT** Only for `type% = 7`. This sets the initial file size in time, which sets the value returned by `MaxTime()` until data written to the file exceeds this time.
- nChan%** Only for `type% = 7`. This sets the number of channels in the file in the range 32 to 400. If you omit `nChan%`, 32 channels are used. Spike2 version 5 can read up to 400 channels, 4 can read up to 100. Version 3 can only read files with 32.
- big%** Only for `type% = 7`. If present this sets the type of the new file. If omitted, a 64-bit `.smrx` file is generated.
- 0 Very old 32-bit `.smr` format with a size limit of 2 GB.
 - 1 Write as a 32-bit `.smr` file with as size limit of 1 TB.
 - 2 Write as a 64-bit `.smrx` file.
- Returns** It returns the view handle (or the handle of the lowest numbered duplicate for a data file with duplicate windows) or a negative error.

See also:

`ChanNew()`, `ChanSave()`, `ChanWriteWave()`, `FileOpen()`, `FileSave()`, `FileSaveAs()`, `FileClose()`, `FilePathSet()`, `MaxTime()`, `MemSave()`, `SampleStart()`, `SampleStop()`, `ViewLink()`, `XYAddData()`, `XYSetChan()`, `XY example`

FileOpen()

This is the equivalent of the File Open menu command. It opens an existing Spike2 data file or a text file in a window, or an external text or binary file. If the file is already opened, a handle for the existing view is returned. The window becomes the new current view. You can create windows as visible or invisible. It is often more convenient to create an invisible window so you can position it before making it visible.

```
Func FileOpen(name$, type% {,mode% {,text$}});
```

- name\$** The name of the file to open. This can include a path. The file name is operating system dependent, see `FileDelete()`. If the name is blank or contains * or ? (Windows only), the file dialog opens for the user to select a file. For file types 8 and 9 only, `name$` can be of the form:
- ```
"Type 1 (*.f11)|*.f11||Type 2 (*.f12;*.f13)|*.f12;*.f13||"
```

This example produces two file types, one with two extensions. There is one vertical bar between the description and the template and two after every template.

- type%** The type of the file to open. The types currently defined are:
- 0 Open a Spike2 data file in a window
  - 1 Open a text file in a window
  - 2 Open an output sequence file in a window
  - 3 Open a Spike2 script file in a window
  - 4 Open a result view file in a window
  - 6 Load configuration file or read configuration from data file (see here for information on file extensions)
  - 8 An external text file without a window for use by `Read()` or `Print()`
  - 9 An external binary file without a window for use by `BRead()`, `BWrite()`, `BSeek()` and other binary routines.
  - 12 Open an XY view file in a window.
- mode%** This optional argument determines how the window or file opens. If the argument is omitted, its value



is 0. For file types 0 to 4 the value is the sum of:

- 1 Make the new window(s) visible immediately. If this flag is not set the window is created, but is invisible.
- 2 Read resource information associated with the file. This may create more than one window, depending on the file type. For data files, it restores the file to the state as it was closed. If the flag is unset, resources are ignored.
- 4 Return an error if the file is already open in Spike2. If this flag is not set and the file is already in use, it is brought to the front and its handle is returned.
- 8 Determines the start folder if the system File Open dialog is used and `name$` does not specify a path. Add 8 to follow the operating system rules, usually opening the last File Open dialog folder. Otherwise open the current folder as set by `FilePathSet()`.

When used with file types 8 and 9 the following values of `mode%` are used. The file pointer (which sets the next output or input operation position) is set to the start of the file in modes 0 and 1 and to the end in modes 2 and 3. You can also add 8, as above, to set where any File Open dialog starts.

- 0 Open an existing file for reading only
- 1 Open a new file (or replace an existing file) for writing (and reading)
- 2 Open an existing file for writing (and reading)
- 3 Open a file for writing (and reading). If the file doesn't exist, create it.

`text$` An optional prompt displayed as part of the file dialog (not supported for `type%=6` on the Macintosh 68k version).

**Returns** If a file opens without any problem, the return value is the view handle for the file (if multiple views open, it is the handle for the first time view created). For configuration files (`type%` of 6), the return value is 0 if no error occurs. If the file could not be opened, or the user pressed Cancel in the file open dialog, the returned value is a negative error code.

If multiple windows are created for a data file, you can get a list of the associated view handles using `ViewList(list%[],64)`.

#### See also:

`FileDelete()`, `FileNew()`, `FileSave()`, `FileSaveAs()`, `FileClose()`, `BRead()`, `BReadSize()`, `BSeek()`, `BWrite()`, `BWriteSize()`, `Read()`, `ViewList()`

## FilePath\$()

This function gets the current directory. This is the place on disk where file open and file save dialogs start. It can also get the path for created data files, the path to the Spike2 application and auto-saving.

**Func** `FilePath$({opt%});`

- `opt%` If omitted this is taken as zero. This determines which directory/folder to get:
- 0 The current directory. This is the directory that incomplete path names (names that do not start with a drive specification or a backslash) are relative to.
  - 1 The path for Spike2 data files created by `FileNew()` as set by Edit menu Preferences in the Sampling tab.
  - 2 The path to the Spike2 application. You should treat this as a read-only location; Spike2 may be installed within the protected Program Files directory tree where non-privileged writes will fail. If you need a location to store data consider using one of the negative `opt%` values.
  - 3 The automatic file naming path as set by the Sampling Configuration Automation tab.

We also support some negative values that return the path to special folders:

- 1 The Desktop folder, for example: `C:\Users\username\Desktop\`
- 2 The users documents folder, for example: `C:\Users\username\Documents\`
- 3 The system folder for Spike2 user and application data, for example: `C:\Users\username\AppData\Local\CED\Spike8\` You can use this location to save files that are specific to the current logon user and Spike2. Stored information should not be Spike2 version specific

unless you also store a version number. You could also consider using the registry with the `Profile()` command for small quantities of information.

- 4 The `Spike8` folder inside the users documents folder (commonly called `My Documents`). It is recommended that you use this location for your data and files if possible. Available from version 7.11.
- 5 The `Spike8Shared` folder inside the documents folder for all users. Available from version 7.11.

Returns A string holding the path or an empty string if an error is detected.

**See also:**

`FileList()`, `FileName$()`, `FilePathSet()`, `Profile()`

## FilePathSet()

This function sets the current directory/folder, and where Spike2 data files created by `FileNew()` are stored until they are sent to their final resting place by `FileSaveAs()`. There are two version of the command. The first sets or optionally creates a directory based on a passed in path, the second prompts the user to choose and optionally create a directory.

```
Func FilePathSet(path${, opt%{, make%}});
Func FilePathSet(path$, opt%, prmpt${, make%});
```

`path$` A string holding the new path to the directory. The path must conform to the rules for path names on the host system and be less than 255 characters long. If the path is empty or a prompt is set, a dialog opens for the user to select an existing directory/folder.

`opt%` If omitted this is taken as zero. This determines which directory/folder to set:

- 0 The current directory. This is the directory that incomplete path names (names that do not start with a drive specification or a backslash) are relative to.
- 1 The path for Spike2 data files created by `FileNew()` as set by Edit menu Preferences in the Sampling tab.
- 2 The path to the application. This is fixed, so an attempt to change it is an error.
- 3 The automatic file naming path as set by the Sampling Configuration Automation tab.

`make%` In the first command version, if this is zero or omitted, all elements of the path must already exist. If set to 1 and `path$` is not empty, the command will create the directory/folder if all elements of the path exist except the last. In the second version of the command that displays a dialog, if this is zero the user can only select an existing path. If set to 1, the user is allowed to create a new directory/folder. If the users sets a path you can find out what was set with `FilePath$()`.

`prmpt$` Optional prompt for use with the dialog. If you supply a prompt, a user dialog will appear using the current value of `path$` as the starting point. If `path$` is empty, the current path set by `opt%` is the starting point.

Returns Zero if the path was set, or a negative error code.

Do not set a path used for creating new files to be a folder that required administrative privilege to access unless you understand the implications of doing this, especially if you intend to allow others to access the file.

**See also:**

`FileList()`, `FileName$()`, `FilePath$()`

## FilePrint()

This function is equivalent to the File menu Print command. It prints some or all of the current view to the printer that is currently set for Spike2. If no printer has been set, the current system printer is used. In a time or result view, it prints a range of data with the x axis scaling set by the display. In a text or log view, it prints a

range of text lines. There is currently no script mechanism to choose a printer; you must do it interactively.

```
Func FilePrint({from{, to{, flags%}}});
```

**from** The start point of the print. This is in seconds in a time view, in bins in a result view and in lines in a text view. If omitted, this is taken as the start of the view.

**to** The end point in the same units as **from**. If omitted, the end of the view is used.

**flags%** 0=portrait, 1=landscape, 2=current setting. If omitted, the current value is used.

**Returns** The function returns 0 if all went well; otherwise it returns a negative error.

The format of the printed output is based on the screen format of the current view. Beware that for time and result views the output could be many (very many) pages long.

All the `FilePrintXXXX()` routines allow you to choose the orientation of the printed output. If you do not set the orientation, the last print orientation you used in the current Spike2 session is set. If you have not printed, the orientation depends on the current orientation set for the system printer.

**See also:**

`FilePrintScreen()`, `FilePrintVisible()`

## FilePrintScreen()

This function is equivalent to the File menu Print Screen command. It prints all visible time, result, XY and text-based views to the current printer on one page. The page positions are proportional to the view positions in the Spike2 application window.

```
Func FilePrintScreen({head${, vTtl%{, box%{, scTxt%{, flags%{, foot$}}}}});
```

**head\$** The page header. If omitted or an empty string, there is no page header.

**vTtl%** Set 1 or higher to print a title above each view, omitted or 0 for no title.

**box%** Set 1 or higher for a box around each view. If omitted, or 0, no box is drawn.

**scTxt%** Set 1 or higher to scale text differently in the x and y directions to match the original. If omitted or 0 scale both directions by the same scale factor.

**flags%** 0=portrait, 1=landscape, 2=current setting. If omitted, the current value is used.

**foot\$** The page footer. If omitted or an empty string, there is no page footer.

**Returns** The function returns 0 if it all went well, or a negative error code.

**See also:**

`FilePrint()`, `FilePrintVisible()`

## FilePrintVisible()

This function prints the current view as it appears on the computer screen to the current printer. In a text window, this prints the lines in the current selection. If there is no selection, it prints the line containing the cursor.

```
Func FilePrintVisible({flags%});
```

**flags%** 0=portrait, 1=landscape, 2=current setting. If omitted, the current value is used.

**Returns** The function returns 0 if all went well; otherwise it returns a negative error.

**See also:**

`FilePrint()`, `FilePrintScreen()`

## FileQuit()

This is equivalent to the File menu Exit command. You are asked if you wish to save any unsaved data before the application closes. If the user cancels the operation (because there were files that needed saving), the script terminates, but the Spike2 application is left running. Use `FileClose(-1, -1)` before `FileQuit()` to guarantee to exit.

```
Proc FileQuit();
```

### See also:

`FileClose()`, `Halt`

## FileSave()

This function is equivalent to the File menu Save command and saves the current view as a file on disk. If the view has not been saved previously the File menu Save As dialog opens to request a file name. It cannot be used with external text or binary files. It cannot be used with a time view unless it has been sampled and has not been saved.

```
Func FileSave();
```

Returns The function returns 0 if the operation was a success, or a negative error code.

### Use with a time view

You cannot use this command for a time view unless it has just been sampled; use `FileSaveAs()` instead. When used with a time view that has just been sampled, if an automatic file name is set, there is no prompt for a file name and the next automatic name is used. However, if automatic filing is enabled, the file is saved as soon as sampling stops, so this function is not needed.

### See also:

`FileOpen()`, `FileSaveAs()`, `FileSaveResource()`, `FileClose()`, `SampleAutoFile()`, `SampleAutoName$()`

## FileSaveAs()

This function saves the current view in its native format, as text, as an external format or as a picture. It is equivalent to the File menu Save As command. This cannot be used for external text or binary files. This command can also name data files created with `FileNew(7, ...)` and after sampling, but read the remarks about time views below. It can also be used with multimedia windows that display video to save the current frame.

```
Func FileSaveAs(name${ ,type%{ ,flags%{ ,text${ ,nCh%|expt%{ ,exp$|big%}}}});
```

**name\$** The output file name including the file extension. If the string is empty or it holds wild card characters \* or ? or **text\$** is not empty, then the File menu Save As dialog opens and **name\$** sets the initial list of files. From version 8.00, if the name is not blank and there is no file extension (the name does not contain a '.' character), Spike2 will add one.

**type%** The type to save the file as (if omitted, type -1 is used):

- 2 Valid for XY and result views only. Opens a File Save As dialog and allows the user to choose the file type to save as.
- 1 Save in the native format for the view. You can also use this to save and name a Spike2 data file immediately after it has been sampled. This will also save any multimedia data associated with the file. The correct sequence is:
  1. Use `FileNew(0, mode%)` to create a data file
  2. Use `SampleStart()` to start data capture
  3. When capture is over use `SampleStop()` to stop capture, or wait for the `SampleStatus()` to indicate that sampling has finished.

4. Use `FileSaveAs(name$, -1)` to set the file name
5. Use `FileClose()` to close the view

This command can also name a file opened with `FileNew(7, ...)`. Once a data file has been named you must use type 0 to save a selection of channels to a new file; you cannot use type -1 again in a time view. Use of this command causes Spike2 to check the file length. Only channels that have been saved to disk are taken into account, so a file that contains only virtual and memory channels will be displayed as if it had zero length after using `FileSaveAs(name$, -1)`.

- 0 Save sections set by `ExportChanList()` to a new Spike2 data file. This will not save associated multimedia files.
- 1 Save the contents of the current view as a text file. If the current view is a time window, Spike2 saves the channels set by `ExportChanList()` in the text format set by `ExportRectFormat()` or `ExportTextFormat()` and `ExportChanFormat()`. See `flag%` for result and XY views.
- 2 Save a text view as an output sequence file.
- 3 Save a text view as a Script file.
- 4 Save as a result view (result views only).
- 5 Save the screen image of a time, result or XY view as a metafile. The file extension sets the format; use `WMF` for a Windows metafile and `EMF` for an enhanced metafile.
- 6 Save sampling configuration in a configuration file. See here for information on file extensions and a possible problem when using old format files.
- 12 Save as XY view (XY views only).
- 13 Save the time, result, XY or multimedia (video only) view as a bitmap. Video frames are saved at the recorded resolution.
- 14 As 13, but saving in Joint Photographic Expert Group (JPEG) format.
- 15 As 13, but saving in Portable Network Graphics (PNG) format.
- 16 As 13, but saving in Tagged Image File Format (TIFF).
- 100 Types 100 upwards identify external exporters installed in the `export` folder (where you will find additional documentation). Time views export the channels and time range set by `ExportChanList()`. Result and XY views export the data set by `flag%`. Codes defined so far:

| Code | Exporter |
|------|----------|
|------|----------|

|     |                                                                                                    |
|-----|----------------------------------------------------------------------------------------------------|
| 100 | MatLab follow this link to see the string format used for the <code>exp\$</code> argument (below). |
|-----|----------------------------------------------------------------------------------------------------|

`flags%` In Spike2 version 7 this argument was called `yes%`. From version 8 it is the sum of the following flags:

- 1 If set, do not ask the user if they want to overwrite an existing file with the same name. Old scripts that set `yes%` to 0 or 1 will see no change in behaviour.
- 8 Determines the start folder if the system File Open dialog is used and `name$` does not specify a path. Add 8 to follow the operating system rules, usually opening the last File Open dialog folder. Otherwise open the current folder as set by `FilePathSet()`. The value 8 is used to match `FileOpen()`.

`text$` An optional prompt displayed as part of the file dialog to prompt the user. The File Save As dialog will appear if `text$` is not empty.

`nCh%` Used when `type%` is 0 to set the number of channels in the exported file in the range 32-400. If omitted, the file has the same number of channels as the source. Spike2 version 4 can read files with up to 100 channels.

`expt%` In Spike2 version 7 this argument was called `flags%`. It is used when saving result and XY views as text or to an external exporter to override the normal behaviour, which is to save all channels and all data points. This optional argument has a default value of 0 and is the sum of:

- 1 Output only visible channels
- 2 Output only selected channels, or visible channels if no channel is selected
- 4 Output only data that is in the visible range (use 5 for visible data)

`exp$` When this is used to export data, this string sets additional export parameters. The string format

depends entirely on the exporter (set by the `type%` argument), so see the exporter documentation for details.

`big%` Used when `type%` is 0 to determine what type of Spike2 data file is created. To force a type set: 0=very old format (.smr) data file limited to 2 GB is size, 1=old format (.smr) with a maximum size of up to 1 TB (for Spike2 version 7), 2=modern 64-bit data file (.smrx). You can also omit this parameter, or set -1 in which case if the file name set ends with .smr we will behave as if 1 were set, otherwise we behave as if 2 were set. If a File Save dialog opens, this field determines which file extensions you can choose.

Returns The function returns 0 if the operation was a success, or a negative error code. Possible errors include:

- 1512 Attempt to save a data file while it is sampling (wait until sampling has stopped)
- 1513 The file exists and is read only
- 1517 The operation was cancelled by the user
- 1520 Attempt to save as an impossible type for example, XY view as a result view
- 1523 The file name is already open in a window in Spike2
- 1542 Some elements of the path do not exist or there are illegal characters in the path
- 1548 An error was detected during the file save operation

### Use with a newly sampled time view

This command can save a time view that has just been sampled as described above for `type%` set to -1. However, if automatic filing is set with `SampleAutoFile()` or by the sampling configuration, the file is saved automatically as soon as sampling finishes and using a `type%` of -1 will generate an error.

When automatic file naming is enabled with `SampleAutoName$()` or with the sampling configuration, you can use this command to override the next sequential name, as long as the file has not already been saved and has finished sampling.

#### See also:

`EditCopy()`, `ExportChanFormat()`, `ExportChanList()`, `ExportTextFormat()`, `ExportRectFormat()`, `FileClose()`, `FileNew()`, `SampleAutoFile()`, `SampleAutoName$()`

## FileSaveResource()

This saves a resource file for the current time, resource or XY view. If a time view is duplicated, resources for all the duplicates are saved.

**Func FileSaveResource({name\$|glob%})**

`name$` The resource file to save to. If the name is "" or contains a "\*" or a "?", the user is prompted for a file. Any extension in the file name is ignored and the extension is set to .s2rx. Remember that \ in a string must be entered as \\ or use /. If the name is "" or contains a "\*" or a "?", the user is prompted for a file.

`glob%` An alternative to `name$`. 0=prompt for a file name, 1=save resources to the resource file associated with the view (this may be a global resource file), 2=save to the global resource file only (if enabled). If both `name$` and `glob%` are omitted, the effect is the same as setting `glob%` to 1.

Returns 0 if all was OK, -1 if the user cancelled the File Save dialog, -2 if the file could not be saved for any other reason.

This command will add information to an existing resource file, or create a new one.

#### See also:

`FileGlobalResource()`, `FileApplyResource()`, `FileOpen()`, `FileSave()`

## FileSize()

This function returns the size of the data file associated with the current time view. During sampling this allows for data that is buffered, but not yet written. This function did not exist before Spike2 version 7.00.

```
Func FileSize();
```

Returns The size of the file, in bytes. This returns an integer (as integers are now 64-bits in size in version 8). In version 7 it returned a real number (as the value returned could exceed integer range). If you want your script to work in both versions 7 and 8 you should assign the result to a real number.

**See also:**

Big files, File size limit

## FileTime\$( )

This function returns a string holding the time at which sampling started. Use `FileTimeDate()` to get the date as numbers. The current view must be a time view. The first 4 arguments are exactly the same as for the `Time$( )` command. Time was not stored in the data file before version 4.03. If there is no time stored, the result is an empty string.

```
Func FileTime$({tBase%{, show%{, amPm%{, sep$ {, ofs%}}}});
```

`tBase%` Specifies the time base to show the time in. You can choose between 24 hour or 12 hour clock mode. If this argument is omitted, 0 is used.

- 0** Operating system settings    2 12 hour format
- 1 24 hour format

`show%` Specifies the time fields to show. Add the values of the required options together and use that as the argument. If this argument is omitted or a value of 0 is used, 7 (1+2+4) is used for 24 hour format and 15 (1+2+4+8) for 12 hour format.

- 1 Show hours                      4 Show seconds
- 2 Show minutes                    8 Remove leading zeros from hours

`amPm%` This sets the position of the “AM” or “PM” string in 12 hour format and has no effect in 24 hour format. If omitted, a value of zero is used. The string that gets printed (“AM” or “PM”) is specified by the operating system.

- 0** Operating system settings    2 Show to the left of the time
- 1 Show to the right of the time 3 Hide the “AM” or “PM” string

`sep$` This string appears between adjacent time fields. If `sep$ = “:”` then the time will appear as 12:04:45. If an empty string is entered or `sep$` is omitted, the operating system settings are used.

`ofs%` If present, this is a time offset, in seconds, to add to the file start time. If omitted or negative it is treated as zero. You can use this to print the time of day at which something happened in the file. If you want to print a time with fractional seconds, you will have to arrange this yourself by adding the fractions to the formatted time.

**See also:**

`Time$( )`, `FileDate$( )`, `FileTimeDate( )`

## FileTimeBase()

This function gets and optionally sets the microseconds per time unit value for the data file associated with the current time view (the clock tick period). Changing the clock tick stretches or shrinks the time axis for all data items in the file. Only the data file header is changed by this operations and you must save the file when you close it or any changes will be lost. *Setting this value will very likely make nonsense of any derived views and*

*Spike2 makes no attempt to sort this out.*

```
Func FileTimeBase({new});
```

**new** If present, and in the range 0.001 to 32767, sets the number of microseconds per clock tick in the file. If the file is sampling, rerunning or write protected, this change has no effect.

**Returns** The microseconds per clock tick at the time of the call.

## An example

Consider a situation where both Spike2 and another system sample data from the same experiment, but the two systems are not synchronised. Given that modern electronics are pretty accurate, we would expect that the two systems would both log data at an accurate clock tick, but that the actual durations of a second as measured on both systems would probably differ, perhaps by as much as one part in 50,000. This is a small difference over 1 second (20 microseconds), but over 30 minutes this would be 36 milliseconds, which might be significant. Let us further assume that this difference stays more or less constant with time, that we can import the data from the other system into a Spike2 data file, and that we have a common marker recorded at the start on both systems and at the end. We want to generate a single file holding the data from both files with all the times synchronised as closely as possible. We do this in the following steps:

1. There is likely to be a time offset between the start markers in both files. This can be eliminated by exporting the data in the files so that the start marker in each file is at time 0. See the `flags%` argument in `ExportChanList()` to do this from a script.
2. Next, work out the time scaling between the two files by looking at the ratio of the end time marker minus the start time marker between the two files. Suppose the time interval is  $t_1$  for file 1 and  $t_2$  for file 2. Unless you are very lucky,  $t_1$  and  $t_2$  will not be the same. To modify file 2 to have the same time interval as file 1 we would make file 2 current and then use:

```
FileTimeBase(FileTimeBase()*t1/t2); 'change the time base
```

3. If you want to amalgamate the two files into one and there are sufficient free channels in one of the files, use `ChanSave()` to move all the channels of one file into free channels in the other. `ChanSave()` adjusts event times and interpolates waveforms to compensate for time base differences between files. If you need to create a file with more free channels, you can do this by exporting one of the files and setting the desired number of channels.

### See also:

`FileNew()`, `ChanSave()`, `ExportChanList()`

## FileTimeDate()

This function returns the time and date at which sampling started as numbers. Use `FileTime$()` and `FileDate$()` to get the result as strings. The current view must be a time view. The arguments are exactly the same as for the `TimeDate()` command. The time and date were not stored in the data file before version 4.03. If there is no information, the returned values are all zero.

```
Proc FileTimeDate(&s%, {&m%, {&h%, {&d%, {&mon%, {&y%, {&wDay%}}}}}});
Proc FileTimeDate(td%[])
```

**s%** If `s%` is the only argument, it is set to the number of seconds since midnight. Otherwise it is set to the number of seconds since the start of the current minute.

**m%** If this is the last argument, it is set to the number of minutes since midnight. Otherwise it is set to the number of full minutes since the start of the hour.

**h%** If present, the number of hours since Midnight is returned in this variable.

**d%** If present, the day of the month is returned as an integer in the range 1 to 31.

**mon%** If present, the month number is returned as an integer in the range 1 to 12.

**y%** If present, the year number is returned here. It will be an integer such as 2002.

**wDay%** If present, the day of the week will be returned here as 0=Monday to 6=Sunday.

**td%[]** If an array is the first and only argument, the first seven elements are filled with time and date data.



The array can be less than seven elements long. Element 0 is set to the seconds, 1 to the minutes, 2 to the hours, and so on.

**See also:**

Date\$(), FileTimeDateSet(), MaxTime(), Seconds(), Time\$(), TimeDate()

## FileTimeDateSet()

This command sets the sampling start time for the file associated with the current time view. This time has been stored in all Spike2 data files since version 4.03. If you create a file using the script language and do not use this command, the sampling start time is set to the time of creation of the file.

```
Func FileTimeDateSet(s%, m%, h%, d%, mon%, y%);
Func FileTimeDateSet(td%[])
```

s% Sets the seconds in the range 0-59.

m% Sets the minutes in the range 0-59.

h% Sets the hours in the range 0-23.

d% Sets the day of the month in the range 1 to 31.

mon% Sets the month of the year in the range 1 to 12.

y% Sets the year in the range 1900 to 2100.

td%[ ] If an array is the first and only argument, the first size elements set the time and date data. Element 0 sets the seconds, 1 sets the minutes, 2 sets the hours, 3 sets the days, 4 sets the month and 5 sets the years. The values should be in the same range as for the s% to y% arguments.

Returns If all the values are in the correct ranges and the file is not read only, the new data is set and the return value is 0. If there is a problem, the return value is a negative error code.

**See also:**

Date\$(), FileTimeDate(), MaxTime(), Seconds(), Time\$(), TimeDate()

## FiltApply()

Applies a set of FIR filter coefficients or a filter in the FIR filter bank to a source waveform or RealWave channel in the current time view and places the result in a destination memory buffer or a disk-based channel. The output can be written as either a waveform or as a RealWave channel.

Each output point is generated from the same number of input points as there are filter coefficients. Half these points are before the output point, and half are after. Where more data is needed than exists in the source file (for example at the start and end of a file and where there are gaps), extra points are made by duplicating the nearest valid point.

```
Func FiltApply(n%|coef[], dest%, srce%, sTime, eTime {,flags%});
```

n% Index of the filter in the filter bank to apply in the range -1 to 11, or

coef[ ] An array holding a set of FIR filter coefficients to apply to the waveform. The size of the passed array sets the number of coefficients.

dest% The channel to hold the filtered waveform: either an unused disk channel, a memory channel with the same sampling frequency as srce% or 0 to create a compatible memory channel and place the filtered waveform in it. When a new channel is created, the channel settings are copied from the old channel.

srce% The source waveform or RealWave channel. There must be at least half the number of sampling coefficients worth of data points before sTime if the output is to start at sTime. Similarly, the channel must extend for the same number of data points beyond eTime if the output is to extend to eTime.

sTime Time to start the output of filtered data. There is no output for areas where there is no input data. If the filter has an even number of coefficients, the output is shifted by half a sample relative to the input.

**eTime** The end of the time range for filtered data.

**flags%** Optional, default value 0. The sum of the following flag values:

- 1 Optimises the destination channel scale and offset values to give the best possible representation of the output as 16-bit integers. For Waveform output channels, this doubles the processing time and existing data in the output channel that is not overwritten is also rescaled. If you do not rescale, the channel's scale and offset are unchanged. However, you run the risk of the output being clipped to the 16-bit range allowed for a waveform channel
- 2 Create a RealWave destination channel in place of a Waveform channel.

**Returns** The channel number that the output was written to or a negative error code. A negative error code is also returned if the user clicks Cancel from the progress bar that may appear during a long filtering operation or if *dest%* is a disk channel that is in use. Delete an existing channel with `ChanDelete(dest%)`.

**See also:**

Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `ChanDelete()`, `FiltAtten()`, `FiltCalc()`, `FiltComment$()`, `FiltCreate()`, `FiltInfo()`, `FiltName$()`, `FiltRange()`

## FiltAtten()

This set the desired attenuation for a filter in the filter bank. When `FiltApply()` or `FiltCalc()` is used, the number of coefficients needed to achieve this attenuation will be generated. A value of zero sets the attenuation back to the default (-65 dB).

```
Func FiltAtten(index%{, dB});
```

**index%** Index of the filter in the filter bank to use in the range -1 to 11.

**dB** If present and negative, this is the desired attenuation for stop bands in the filter.

**Returns** The desired attenuation for a filter at the time of the call.

**See also:**

Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply()`, `FiltCalc()`, `FiltComment$()`, `FiltCreate()`, `FiltInfo()`, `FiltName$()`, `FiltRange()`

## FiltCalc()

The calculation of filter coefficients can take an appreciable time; this routine forces the calculation of a filter for a particular sampling frequency if it has not already been done. If you do not force the calculation, you can still use `FiltApply()` to apply a filter. However, the coefficient calculation will then be done at the time of filter application, which may not be desirable if the filtering operation is time critical.

```
Func FiltCalc(index%, sInt{, coeff[]{, &dBGot {, nCoef%}}});
```

**index%** Index of the filter in the filter bank to use in the range -1 to 11.

**sInt%** The sample interval of the waveform you are about to filter. This is the value returned by `BinSize()` for a waveform channel.

**coeff** An array to be filled with the coefficients used for filtering. If the array is too small, as many elements as will fit are set. The maximum size needed is 2047 (but most filters require far fewer coefficients).

**dBGot** If present, set to the attenuation in negative dB attained by the filter coefficients.

**nCoef%** If present, sets the number of coefficients used in the calculation (use an even number for a full differentiator and an odd number for all other filter types).

**Returns** The number of coefficients generated by the filter. If you are passing the result to `FiltApply()`, use `coeff[:n]`, not `coeff[]` to pass *n* coefficients.

## An example

Suppose the first filter in the bank (index 0) is a low pass filter with the pass band edge at 50 Hz. If we know that we will need to filter a channel 4 (sampled at 200 Hz) with this filter, we may want to calculate the coefficients needed in advance:

```
FiltCalc(0, BinSize(4));
```

This will calculate a filter corresponding to the specification of filter 0 for a sampling frequency of 200 Hz with an attenuation in the stop band of at least the current desired attenuation value for this filter.

## Constraints on filters

The calculation of coefficients is a complex process and can produce silly results due to floating point rounding errors in some situations. To ensure that you will always get a useful result there is a limit to how small and how big a transition gap can be, relative to the sampling frequency. There is a similar limit on the width of a pass or stop band:

- The transition gap and the width of a pass or stop band cannot be smaller than 0.0025 of the sampling frequency.
- The transition gap cannot be larger than 0.12 of the sampling frequency.

This function always calculates a set of coefficients, but may alter the filter specification in order to do it (these changes are temporary, see later). This can happen in two cases:

1. If the sampling frequency is such that to produce the filter, the transition gap and/or pass and stop band widths are outside their limits, then the widths are set to the limits before calculating the filter. In our 50 Hz low pass filter example, if we calculate it with respect to a 12 kHz sampling frequency, the minimum pass band width is  $12000 * 0.0025 = 30$  Hz. So, the filter would be changed to a 60 Hz low pass filter.
2. If half the sampling frequency (the Nyquist frequency) is less than an edge of a pass or stop band, certain attributes of the filter are lost. In our 50 Hz low pass filter example, if we tried to calculate with a sampling frequency of 80 Hz, we would see that the Nyquist frequency is 40 Hz. No frequency above 40 Hz can be represented in a waveform sampled at 80 Hz, so a 50 Hz low pass filter is equivalent to an “All pass” filter. The filter specification will be altered to reflect this before calculating.

Any changes made to a filter specification to accommodate a particular calculation are made with reference to the original specification, not the specification that was last used for a calculation.

### See also:

Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply()`, `FiltAtten()`, `FiltComment$()`, `FiltCreate()`, `FiltInfo()`, `FiltName$()`, `FiltRange()`

## FiltComment\$()

This function gets and sets the comment associated with a filter in the filter bank.

```
Func FiltComment$(index% {, new$});
```

`index%` Index of the filter in the filter bank to use in the range -1 to 11.

`new$` If present, sets the new comment.

Returns The previous comment for the filter at the index.

### See also:

Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply()`, `FiltAtten()`, `FiltCalc()`, `FiltCreate()`, `FiltInfo()`, `FiltName$()`, `FiltRange()`

## FiltCreate()

This function creates a FIR filter in the filter bank to the supplied specification and gives it a standard name and comment.

```
Func FiltCreate(index%, type%{, trW{, edge1{, edge2{, ...}}});
```

**index%** Index of the new filter in the filter bank in the range -1 to 11. This action replaces any existing filter at this index.

**type%** The type of the filter desired (see table).

**trW** The transition width of the filter. This is the frequency interval between the edge of a stop band and the edge of the adjacent pass band.

**edgeN** This is a list of edges of pass bands in Hz. See the table.

Returns 0 if there was no problem or a negative error code if the filter was not created.

This table shows the relationship between different filter types and the meaning of the corresponding arguments. The numbers in brackets indicate the nth pass band when there is more than 1. An empty space in the table means that the argument is not required.

| type% |                         | trW | edge1   | edge2   | edge3   | edge4   |
|-------|-------------------------|-----|---------|---------|---------|---------|
| 0     | All stop                |     |         |         |         |         |
| 1     | All pass                |     |         |         |         |         |
| 2     | Low pass                | Yes | High    |         |         |         |
| 3     | High pass               | Yes | Low     |         |         |         |
| 4     | Band pass               | Yes | Low     | High    |         |         |
| 5     | Band stop               | Yes | High(1) | Low(2)  |         |         |
| 6     | Low pass differentiator | Yes | High    |         |         |         |
| 7     | Differentiator          |     |         |         |         |         |
| 8     | 1.5 Band Low pass       | Yes | High(1) | Low(2)  | High(2) |         |
| 9     | 1.5 Band High pass      | Yes | Low(1)  | High(1) | Low(2)  |         |
| 10    | 2 Band pass             | Yes | Low(1)  | High(1) | Low(2)  | High(2) |
| 11    | 2 Band stop             | Yes | High(1) | Low(2)  | High(2) | Low(3)  |

The values entered correspond to the text fields shown in the Filter edit dialog box.

**See also:**

Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply()`, `FiltAtten()`, `FiltCalc()`, `FiltComment$()`, `FiltInfo()`, `FiltName$()`, `FiltRange()`

## FiltInfo()

Retrieves information about a filter in the bank.

```
Func FiltInfo(index%{, what%});
```

**index%** Index of the filter in the filter bank to use in the range -1 to 11.

**what%** Which bit of information about the filter to return:

- 2 Maximum what% number allowed
- 1 Desired attenuation
- 0 type (if you supply no value, 0 is assumed)
- 1 Transition width
- 2-5 edge1-edge4 given in `FiltCreate()`

Returns The information requested as real number.

**See also:**

Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply()`, `FiltAtten()`, `FiltCalc()`, `FiltComment$()`, `FiltCreate()`, `FiltName$()`, `FiltRange()`

## FiltName\$()

This function gets and/or sets the name of a filter in the filter bank.

```
Func FiltName$(index%, new$);
```

`index%` Index of the filter in the filter bank to use in the range -1 to 11.

`new$` If present, sets the new name.

Returns The previous name of the filter at that index.

### See also:

Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply()`, `FiltAtten()`, `FiltCalc()`, `FiltComment$()`, `FiltInfo()`, `FiltCreate()`, `FiltRange()`

## FiltRange()

Retrieves the minimum and maximum sampling rates that this filter can be applied to without the specification being altered. See the `FiltCalc()` command, Constraints on filters for more information.

```
Proc FiltRange(index%, &minFr, &maxFr);
```

`index%` Index of the filter in the filter bank to use in the range -1 to 11.

`minFr` Returns the minimum sampling frequency you can calculate the filter with respect to, so that no transition width is greater than the maximum allowed and that no attributes of the filter are lost.

`maxFr` Returns the maximum sampling frequency you can calculate the filter with respect to without the transition (or band) widths being smaller than allowed.

It is possible to create a filter that cannot be applied to any sampling frequency without being changed. This will be apparent because `minFr` will be larger than `maxFr`.

### See also:

Interactive FIR filtering, Filter banks, Filter types, Technical details of FIR filters, `FiltApply()`, `FiltAtten()`, `FiltCalc()`, `FiltComment$()`, `FiltInfo()`, `FiltCreate()`, `FiltName$()`

## FIRMake()

This function creates FIR filter coefficients and places them in an array ready for use by `ArrFilt()`. This command is very similar in operation to the DOS program `FIRMake` and has similar input requirements. Unless you need precise control over all aspects of filter generation, you may find it easier to use `FiltCalc()` or `FIRQuick()`. You will need to read the detailed information about FIR filters in the description of the Digital Filter dialog to get the best results from this command.

```
Proc FIRMake(type%, param[][] , coef[] {, nGrid {, extFr[]}});
```

`type%` The type of filter file to produce: 1=Multiband filter, 2=Differentiator, 3=Hilbert transformer, 4=Multiband pink noise (Multiband with 3 dB per octave roll-off).

`param` This is a 2-dimensional array. The size of the first dimension must be 4 or 5. The size of the second dimension (`n`) should be the number of bands in your filter. You pass in 4 values for each band (indices 0 to 3) to describe your filter:

Indices 0 and 1 are the start and end frequency of each band. All frequencies are given as fraction of a sampling frequency and so are in the range 0 to 0.5.

Index 2 is the function of the band. For all filter types except a differentiator, this is the gain of the filter in the band in the range 0 to 1; the most common values are 0 for a stop band and 1 for a pass band. For a differentiator, this is the slope of the filter in the band, normally not more than 2. The gain at any frequency  $f$  in the band is given by  $f^{\text{function}}$ .

Index 3 is the relative weight to give the band. The weight sets the relative importance of the band in

multiband filters. The program divides each band into frequency points and optimises the filter such that the maximum ripple times the weight in each band is the same for all bands. The weight is independent of frequency, except in the case of the differentiator, where the weight used is weight/frequency.

If there is an index 4 (the size of the first dimension was 5), this index is filled in by the function with the ripple in the band in dB.

- coef** An array into which the FIR filter coefficients are placed. The size of this array determines the number of filter coefficients that are calculated. It is important, therefore, to make sure this array is exactly the size that you need. The maximum number of coefficients is 2047.
- nGrid** The grid density for the calculation. If omitted or set to 0, the default density of 16 is used. This sets the density of test points in internal tables used to search for points of maximum deviation from the filter specification. The larger the value, the longer it takes to compute the filter. There is seldom any point changing this value unless you suspect that the program is missing the peak points.
- extFr** An array to hold the list of extremal frequencies (the list of frequencies within the bands which have the largest deviation from the desired filter). If there are  $n\%$  coefficients, there are  $(n\%+1)/2$  extremal frequencies.

The parameters passed in must be correct or a fatal error results. Errors include: overlapping band edges, band edges outside the range 0 to 0.5, too many coefficients, differentiator slope less than 0. If the filter is not a differentiator the band function must lie between 0 and 1, the band weight must be greater than 0.

For example, to create a low pass filter with a pass band from 0 to 0.3 and a stop band from 0.35 to 0.5, and no return of the ripple, you would set up `param` as follows:

```
var param[4][2] 'No return of ripple, 2 bands
para[0][0] := 0; 'Starting frequency of pass band
para[1][0] := 0.3; 'Ending frequency of pass band
para[2][0] := 1; 'Desired gain (unity)
para[3][0] := 1; 'Give this band a weighting of 1
para[0][1] := 0.35; 'Starting frequency of stop band
para[1][1] := 0.5; 'Ending frequency of stop band
para[2][1] := 0; 'Desired gain of 0 (stop band)
para[3][1] := 10; 'Give this band a weighting of 10
```

### See also:

More about `FIRMake()` filter types, Interactive FIR filtering, Filter banks, Technical details of FIR filters, `ArrFilt()`, `FiltApply()`, `FiltCalc()`, `FIRQuick()`, `FIRResponse()`

## FIRQuick()

This function creates a set of filter coefficients in the same way the `FIRMake()` does, but many of the parameters are optional, allowing the most common filters to be created with a minimal specification.

```
Func FIRQuick(coef[], type%, freq {, width {, atten}});
```

- coef** An array into which the FIR filter coefficients are placed. The size of this array should be 2047 (was 255 in versions before 4.01). This is the maximum number of coefficients that can be created and this function reserves the right to return as many as it feels necessary up to that value to create a good filter.
- type%** This sets the type of filter to create. 0=Low pass, 1=High pass, 2=Band pass, 3=Band stop and 4=Differentiator.
- freq** This is a fraction of the sampling rate in the range 0 to 0.5 and means different things depending on the type of filter:
- For Low pass, High pass and Differentiator types, this represents the cut-off frequency. This is the frequency of the higher edge of the first frequency band.
- For Band pass and Band stop filters, this is the midpoint of the middle frequency band: the pass band in a Band pass filter, the stop band in a Band stop filter.
- width** For Low pass, High pass and Differentiator filters, this is the width of the transition gap between the stop band and the pass band. The default value is 0.02 and there is an upper limit of 0.1 on this argument.

For a Band pass, or Band stop filter, `width` is the width of the middle band. E.g. if you ask for a Pass band filter with the `freq` parameter to be 0.25 and the width to be 0.05, the middle pass band will be from 0.2 to 0.3. For these types of filter, you still need a positive transition width. This transition width is 0.02 and cannot be changed by the user.

`atten` The desired attenuation in the stop band in dB. The default is 50 dB. This is analogous to the desired attenuation in the `FiltAtten()` command.

Returns The number of coefficients calculated. If the array is not large enough the coefficient list is truncated (and the result is useless). If you are passing the result to `FiltApply()`, use `coef[:n]`, not `coef[]` to pass `n` coefficients.

### See also:

More about `FIRMake()` filter types, Interactive FIR filtering, Filter banks, Technical details of FIR filters, `ArrFilt()`, `FiltApply()`, `FiltCalc()`, `FIRMake()`, `FIRResponse()`

## FIRResponse()

This function retrieves the frequency response of a FIR filter as amplitude or in dB given the filter coefficients.

```
Proc FIRResponse(resp[], coef[]{, as%{, type%}});
```

`resp` The array to hold the frequency response. This array will be filled regardless of its size. The first element is the amplitude response at 0 Hz and the last is the amplitude response at the Nyquist frequency. The remaining elements are set to the response at a frequency proportional to the element position in the array.

`coef` The coefficient array calculated by `FIRMake()`, `FIRQuick()` or `FiltCalc()`.

`as%` If this is 0 or omitted, the response is in dB (0 dB is unchanged amplitude), otherwise as linear amplitude (1.0 is unchanged).

`type%` If present, sets the filter type. The types are the same as those supplied for `FIRQuick()`: 0=Low pass, 1=High pass, 2=Band pass, 3=Band stop and 4=Differentiator. If a type is given, the time to calculate the response is halved. If you are not sure what type of filter you have, or you have a type not covered by the `FIRQuick()` types, then omit `type%` or set it to -1.

## FitCoef()

This command gives you access to the fit coefficients for the next `FitData()` fit. You can return the values from any type of fit and set the initial values and limits and hold values fixed for iterative fits. There are two command variants:

### Set and get coefficients

This command variant lets you read back the current coefficient values and set the coefficient values and limits for iterative fitting:

```
Func FitCoef({num%{, new{, lower{, upper}}});
```

`num%` If this is omitted, the return value is the number of coefficients in the current fit. If present, it is a coefficient number. The first coefficient is number 0. If `num%` is present, the return value is the coefficient value for the existing fit, or if there is no fit, the coefficient value that would be used as the starting point for the next iterative fit is returned.

`new` If present, this sets the value of coefficient `num%` for the next iterative fit.

`lower` If present, this sets the lower limit for coefficient `num%` for the next iterative fit. There is currently no way to read back the coefficient limits. There is also no check made that the limits are set to sensible values.

`upper` If present, this sets the upper limit for coefficient `num%` for the next iterative fit.

Returns The number of coefficients or the value of coefficient `num%`.

## Get and set the hold flags

This command variant allows you to hold some coefficients at their current values during the next fit.

**Func FitCoef(hold%[]);**

**hold%** An array of integers to correspond with the coefficients. If the array is too long, extra elements are ignored. If it is too short, extra coefficients are not affected. Set `hold%[i%]` to 1 to hold coefficient `i%` and to 0 to fit it. If `hold%[i%]` is less than 0, the hold state is not changed, but `hold%[i%]` is set to 1 if the corresponding coefficient is held and to 0 if it is not held.

Returns This always returns 0.

### See also:

`FitData()`, `FitValue()`, `FitExp()`, `ChanFitCoef()`, More about curve fitting

## FitData()

This function, together with `FitCoef()` and `FitValue()`, lets you apply the same fitting functions that are available for channel data to data in arrays. You supply arrays of x and y data points and an optional array holding the standard deviation of the input data point y values. There are three command variants:

### Initialise fit information

The first variant sets the type of fit. If you select an iterative fit, the initial values of the fitting coefficients are reset to standard values and any "hold" flags set by `FitCoef()` are cleared. You can set your own initial values with the `FitCoef()` command or make a guess at the initial values when performing the fit.

**Func FitData(type%, order%);**

**type%** The fit type. 0=Clear any fit, 1=Exponential, 2=Polynomial, 3=Gaussian, 4=Sine, 5=Sigmoid.

**order%** If positive, this is the order of the fit, if negative it is minus the number of fitted coefficients. See the information about each fit for the allowed values for each fit type. If `type%` is 0 this argument is ignored and should be 0.

Returns The number of fit coefficients for the fit or a negative error code.

### Exponential fit

This fits multiple exponentials by an iterative method. The data is fitted to the equation:

$$y = a_0 \exp(-x/a_1) + a_2 \exp(-x/a_3) \dots \quad \text{For an even number of coefficients}$$

$$y = a_0 \exp(-x/a_1) + a_2 \exp(-x/a_3) \dots + a_n \quad \text{For an odd number of coefficients}$$

You can set up to 10 coefficients or orders 1 to 5. If you use a fit order, the number of coefficients is the order times 2. See the `FitExp()` command for more information. Coefficient estimates are effective for orders 1 and 2.

### Polynomial fit

This fits  $y = a_0 + a_1x + a_2x^2 + a_3x^3 \dots$  to a set of (x,y) data points. The fitting is by a direct method; there is no iteration. The fit order is the highest power of x to fit in the range 1 to 10. The number of coefficients is the fit order plus 1.

### Gaussian fit

This fits multiple Gaussians by an iterative method. The data is fitted to the equation:

$$y = a_0 \exp(-1/2(x-a_1)^2/a_2^2) + a_3 \exp(-1/2(x-a_4)^2/a_5^2) + \dots$$

The fitted parameters (coefficients) are the  $a_i$ . You can fit up to 3 Gaussians (order 1 to 3). The number of coefficients is given by the fit order times 3. Coefficient estimates become less useful as the order increases.



## Sine fit

This fits multiple sinusoids by an iterative method. The data is fitted to the equation:

$$y = a_0 \sin(a_1 x + a_2) + a_3 \sin(a_4 x + a_5) + \dots \{+ a_{3n}\}$$

The fitted coefficients are the  $a_i$ . Angles are evaluated in radians. You can fit up to 3 sinusoids (order 3) and an optional offset. Although the function is given in terms of sine functions, you can easily convert to cosines by subtracting  $\pi/2$  from the phase angle ( $a_2, a_5, a_8$ ) after the fit. The coefficient count can be set to 3, 4, 6, 7, 9 or 10. If you use orders, the number of coefficients is order times 3 and you cannot set an offset. A useful coefficient estimate is made for a single sinusoid fit.

## Sigmoid fit

This fits a single Boltzmann sigmoid by an iterative method. The data is fitted to the equation:

$$y = a_0 + (a_1 - a_0) / (1 + \exp((a_2 - x) / a_3))$$

In terms of the fitted result,  $a_0$  and  $a_1$  are the low and high fitting limits,  $a_2$  is the X50 point and  $a_3$  is related to the reciprocal of the slope at the X50 point. The slope at the X50 point, where  $x$  is  $a_2$ , is  $(a_1 - a_0) / (4 a_3)$ . You can set order 1 only, or 4 coefficients.

## Perform the fit

This variant of the command does the fit set by the previous variant. Use the `FitCoef()` command to preset fit coefficients and to read back the result of the fit.

```
Func FitData(opt%, y[], x[][, s[]|s{, &err{, maxI% {,&iTer%{, covar[]
[]}]});
```

`opt%` 1=Estimate the coefficients before fitting, 0=use current values. Note that the estimates are usually only useful for a small number of coefficients.

`y[]` An array of y values to be fitted.

`x[]` A corresponding array of x values.

`s[]|s` A corresponding array of standard deviations for the data points defined by `y[]` and `x`, or a single value, being the standard deviation of each point. If this value is omitted or set to 1.0, the result is a least squares fit. If standard deviations are supplied, the result is a chi-squared fit.

`err` If present, this optional variable is updated with the chi-squared or least-squares error between the fit and the data.

`maxI%` If present, this changes the maximum number of iterations from 100.

`iTer%` If present, this integer variable is updated with the count of iterations done.

`covar` An optional two dimensional array of size at least `[nCoef][nCoef]` that is returned holding the covariance matrix when the fit is complete. It is changed if the return value is -1, 0 or 1. However, the values it contains are probably not useful unless the return value is 0.

Returns 0 if the fit is complete, 1 if max iterations done, or a negative error code: -1=the fit is not making progress (results may be OK), -2=the fit failed due to a singular matrix, -5=the fit caused a floating point error, -6=too little data for the number of coefficients, -7=unknown fitting function, -8=ran out of memory during the fit (too many data points), -9=the fit was set up with bad parameters.

## Get fit information

This variant of the command returns information about the current fit.

```
Func FitData({opt%});
```

`opt%` This determines what to return. `opt%` values match `ChanFit()`, where possible. The returned information for each value of `opt%` is:

| <code>opt%</code> | Returns                         | <code>opt%</code> | Returns               |
|-------------------|---------------------------------|-------------------|-----------------------|
| 0                 | Fit type of next fit            | 1                 | Fit order of next fit |
| -1                | 1=a fit exists, 0=no fit exists | -8                | Not used              |

|    |                             |     |                               |
|----|-----------------------------|-----|-------------------------------|
| -2 | Type of last fit or 0       | -9  | Not used                      |
| -3 | Number of coefficients      | -10 | Not used                      |
| -4 | Chi or least-squares error  | -11 | 1=chi-square, 0=least-square  |
| -5 | Fit probability (estimated) | -12 | Last fit result code          |
| -6 | Lowest x value fitted       | -13 | Number of fitted points       |
| -7 | Highest x value fitted      | -14 | Number of fit iterations used |

Returns The information requested by the `opt%` argument or 0 if `opt%` is out of range.

## FitExp()

This command fits multiple exponentials to arrays of x,y data points, with an optional weight for each point. Fitting is by an iterative method. The data is fitted to the equation:

$$y = a_0 \exp(-x/a_1) + a_2 \exp(-x/a_3) \dots \quad \text{For an even number of coefficients}$$

$$y = a_0 \exp(-x/a_1) + a_2 \exp(-x/a_3) \dots + a_n \quad \text{For an odd number of coefficients}$$

The fitted coefficients are the  $a_i$ ; odd numbered  $a_i$  are assumed to be positive. You can fit up to 5 exponentials or 4 exponentials and an offset. However, experience shows that trying to fit more than two exponentials requires care. The fit from even two exponentials should be viewed with caution, especially if the odd coefficients are similar. The commands to implement this are:

### Set up the problem

The first command sets the number of exponentials to fit and the data set to be fitted. You must call this function before you call any of the others.

```
func FitExp(nCoef%, y[], x[]{, s[]|s});
```

`nCoef%` The number of coefficients to fit in the range 2 to 10. If this is even, the first form of the function above is used. If it is odd, the final coefficient is an offset.

`y[]` An array of y data values. The length of the array must be at least `nCoef%`.

`x[]` An array of x data values. The length of the array must be at least `nCoef%`.

`s` An optional argument which is either an array with one value for each y data point or a single value for all data points. If the value is the standard deviation, then the error value returned when you iterate to find the best fit is the chi-squared value and the fit is a chi-squared fit.

If this value is proportional to the error in the y values, then the fit is still a chi-squared fit, and the error returned is proportional to the chi-squared value. If you omit this array, the fit is a least-squares error fit, and the error value returned is the sum of the squares of the errors in the y values.

Returns The function returns 0. There is no other return value as all errors stop the script.

The number of data points is set by the smallest of the sizes of the `y[]`, `x[]` or `s[]` (if present) arrays. The number of data points must be at least the number of coefficients.

### Set coefficient values and ranges

This variant of the function sets the initial value of each coefficient and optionally sets the range of allowed values. You can call this function at any time after the setup call and before the iterate call (below) has returned 0 indicating that the fit is completed.

It is important that you give reasonable initial values for the coefficients, especially when you fit more than one exponent. You should limit the odd coefficient values (the time constants) so that they cannot be zero and make sure that multiple exponents do not have overlapping ranges. If two exponents get similar values, the fit is degenerate and will wander around forever without getting anywhere. However, setting too rigid a range may damage the fitting process as sometimes the minimisation process has to follow a convoluted n-dimensional path to reach the goal, and the path may need to wander quite a bit. Let experience be your guide.

If you do not give starting values, the command will make a simplistic guess at the fitting values. As we expect that you know more about the "right" answer than the command does, we suggest that you set the values you want.

```
func FitExp(coef%, val{, lo, hi});
```

**coef%** The coefficient to set. The first coefficient is number 0, the last is `nCoef%-1`.

**val** The initial value to assign to the coefficient. If you have set low and high limits, and the value is outside these limits, it is set to the nearer limit.

**lo,hi** If present, these two values set the acceptable range of values for this coefficient. If omitted, or if both values are set to the same value, there is no limit. The value of the coefficient is tested against these limits after each iterative step, and if it exceeds a limit, it is set to that limit.

### Iterate to a solution

Once you have set up the problem and given initial values to your coefficients, you can start the iteration process that will move the coefficients from their starting values to new values that minimise the error (optionally scaled by the `s` argument).

```
func FitExp(a[], &err{, maxI%{, &iTer%{, covar[][]}});
```

**a[]** An array of size at least `nCoef%` that is returned holding the current set of coefficient values. The first amplitude is in `a[0]`, the first exponent in `a[1]`, the second amplitude in `a[2]`, the second exponent in `a[3]` and so on.

**err** A real variable returned as the sum over the data points of  $(yx[i]-y[i])^2/s[i]^2$  if `s[]` is used or holding the sum of  $(yx[i]-y[i])^2$  if `s[]` is not used, where `yx[i]` is the value predicted from the coefficients at the `x` value `x[i]`.

**maxI%** This is the maximum number of fitting iterations to do before returning from the function. If you omit this value, the function sets 1000. You can set any value from 1 to 10000. If you set more than 10000, the number is limited to 10000.

**iTer%** An optional integer variable that is returned holding the number of iterations done before the function returned.

**covar** An optional two dimensional array of size at least `[nCoef%][nCoef%]` that is returned holding the covariance matrix when the fit is complete. It is not changed unless the function return value is 0 or -1.

Returns 1 if the number of iterations are exhausted and the fit has not converged, 0 if the fit has converged, -1 if the fit is not improving but the result may be OK. Other negative numbers indicate failure.

### Select coefficients to fit

Normally the command fits all the coefficients, but you can use this variant to select the coefficients to fit. You can use this command at any time after you have set the problem until the iteration variant returns 0 or -1.

```
func FitExp(fit%[]);
```

**fit%[]** An array of at least `nCoef%` integers. If `fit%[i]` is 0, coefficient `i` is held constant, otherwise it is fitted. If all elements are 0, all arguments are fitted.

The effect of this command persists until either the iteration variant returns a value less than 1, or you set up a new problem, or you call this variant again.

### An example

This is a template for using these commands to fit all the coefficients:

```
const nData%:=50; 'set number of data elements
var x[nData%], y[nData%]; 'space for our arrays
var coefs[4],err; 'coefficients and squared error
... 'in here goes code to get the data
FitExp(4, y[], x[]); 'fit two exponentials (no sigma array)
FitExp(0, 1.0, 0.2, 4); 'set first amplitude and limit range
FitExp(1, .01, .001, .03); 'set first time constant and range
FitExp(2, 2.0, 0.1, 6); 'set second amplitude and limit range
FitExp(3, .08, .03, .15); 'set second time constant and range
repeat
 DrawMyData(coefs[], x[], y[]); 'A function you write to show progress
until FitExp(coefs[], err, 1) < 1;
DrawMyData(coefs[], x[], y[]); 'Show the final state
```

**See also:**

More about curve fitting, `ChanFit()`, `FitGauss()`, `FitLinear()`, `FitNLUser()`, `FitPoly()`, `FitSigmoid()`, `FitSin()`

**FitGauss()**

This command fits multiple Gaussians to x,y data points with an optional weight for each point. Fitting is by an iterative method. The input data is fitted to the equation:

$$y = a_0 \exp(-1/2(x-a_1)^2/a_2^2) + a_3 \exp(-1/2(x-a_4)^2/a_5^2) + \dots$$

The fitted parameters (coefficients) are the  $a_i$ . You can fit up to 3 Gaussians. The commands to implement this are:

**Set up the problem**

The first command sets the number of Gaussians to fit and the data set to be fitted. You must call this function before you call any of the others.

```
func FitGauss(nCoef%, y[], x[]{, s[]|s});
```

**nCoef%** The number of coefficients to fit. The legal values are 3, 6 and 9 for one, two and three Gaussians.

**y[]** An array of y data values. The length of the array must be at least **nCoef%**.

**x[]** An array of x data values. The length of the array must be at least **nCoef%**.

**s** An optional argument which is either an array with one value for each y data point or a single value for all data points. If the value is the standard deviation, then the error value returned when you iterate to find the best fit is the chi-squared value and the fit is a chi-squared fit.

If this value is proportional to the error in the y values, then the fit is still a chi-squared fit, and the error returned is proportional to the chi-squared value. If you omit this array, the fit is a least-squares error fit, and the error value returned is the sum of the squares of the errors in the y values.

Returns The function returns 0. There is no other return value as all errors stop the script.

The number of data points is set by the smallest of the sizes of the **y[]**, **x[]** or **s[]** (if present) arrays. The number of data points must be at least the number of coefficients. If it is not, you will get a fatal error, so check this before calling the function.

**Set coefficient values and ranges**

This variant of the function sets the initial value of each coefficient and optionally sets the range of allowed values. You can call this function at any time after the setup call and before the iterate call (below) has returned 0 indicating that the fit is completed.

In this type of fitting it is very important that you give reasonable starting values for the coefficients. In particular, when fitting multiple Gaussians, it is usual that the centre of each distribution is easy to determine. If you can set the centres and limit them so that they cannot overlap, the fit usually will proceed without any problems, even for multiple Gaussians. If you do not give starting values, the command will make a simplistic guess at the fitting values. As we expect that you know more about the “right” answer than the command does, we suggest that you set the values you want.

```
func FitGauss(coef%, val{, lo, hi});
```

**coef%** The coefficient to set. The first coefficient is number 0, the last is **nCoef%-1**.

**val** The initial value to assign to the coefficient. If you have set low and high limits, and the value is outside these limits, it is set to the nearer limit.

**lo, hi** If present, these two values set the acceptable range of values for this coefficient. If omitted, or if both values are set to the same value, there is no limit. The value of the coefficient is tested against these limits after each iterative step, and if it exceeds a limit, it is set to that limit.

As long as you make a reasonable estimate of the centre points, there should be no problems fitting multiple Gaussians.

## Iterate to a solution

Once you have set up the problem and given initial values to your coefficients, you can start the iteration process that will move the coefficients from their starting values to new values that minimise the error (optionally scaled by the `s` argument).

```
func FitGauss(a[], &err{, maxI{, &iTer{, covar[][]}});
```

`a[]` An array of size at least `nCoef%` that is returned holding the current set of coefficient values. The first amplitude is in `a[0]`, the first centre in `a[1]`, the first sigma in `a[2]`, the second amplitude in `a[3]` and so on.

`err` A real variable returned as the sum over the data points of  $(yx[i]-y[i])^2/s[i]^2$  if `s[]` is used or holding the sum of  $(yx[i]-y[i])^2$  if `s[]` is not used, where `yx[i]` is the value predicted from the coefficients at the `x` value `x[i]`.

`maxI%` This is the maximum number of fitting iterations to do before returning from the function. If you omit this value, the function sets 1000. You can set any value from 1 to 10000. If you set more than 10000, the number is limited to 10000.

`iTer%` An optional integer variable that is returned holding the number of iterations done before the function returned.

`covar` An optional two dimensional array of size at least `[nCoef%][nCoef%]` that is returned holding the covariance matrix when the fit is complete. It is not changed unless the function return value is 0 or -1.

Returns 1 if the number of iterations are exhausted and the fit has not converged, 0 if the fit has converged, -1 if the fit is not improving but the result may be OK. Other negative numbers indicate failure.

Remember that even when a minimum is found, there is no guarantee that this is the minimum. It is the best minimum that this algorithm can find given the starting point.

## Select coefficients to fit

Normally the command fits all the coefficients, but you can use this variant to select the coefficients to fit. You can use this command at any time after you have set the problem until the iteration variant returns 0 or -1.

```
func FitGauss(fit%[]);
```

`fit%[]` An array of at least `nCoef%` integers. If `fit%[i]` is 0, coefficient `i` is held constant, otherwise it is fitted. If all elements are 0, all arguments are fitted.

The effect of this command persists until either the iteration variant returns a value less than 1, or you set up a new problem, or you call this variant again.

## An example

This is a template for using these commands to fit all the coefficients:

```
const nData%:=50; 'set number of data elements
var x[nData%], y[nData%]; 'space for our arrays
var s[nData%]; 'space for sigma of each point
var coefs[4], err; 'coefficients and error squared
... 'in here goes code to get the data
FitGauss(3, y[], x[], s[]); 'fit one gaussian
FitGauss(0, 1.0, 0.2, 4); 'set amplitude and limit range
FitGauss(1, 2, 1.5, 2.5); 'set centre of the gaussian and range
FitGauss(2, 0.5, 0.3, 1.9); 'set width and limit range
repeat
 DrawMyData(coefs[], x[], y[]); 'Some function to show progress
until FitGauss(coefs[], err, 1) < 1;
DrawMyData(coefs[], x[], y[]); 'Show the final state
```

## See also:

More about curve fitting, `ChanFit()`, `FitExp()`, `FitLinear()`, `FitNLUser()`, `FitPoly()`, `FitSigmoid()`, `FitSin()`

## FitLine()

This function calculates the least-squares best-fit line to a set of data points from a time view waveform, RealWave or WaveMark channel or a result view. See ChanFit() for XY views. It fits the expression:

$$y = \mathbf{m}x + \mathbf{c}$$

through the data points  $(x_i, y_i)$  so as to minimise the error given by:

$$\text{Sum}_i(y_i - \mathbf{m}x_i - \mathbf{c})^2$$

In this expression,  $\mathbf{m}$  is the gradient of the line and  $\mathbf{c}$  is the y axis intercept when x is 0.

```
Func FitLine(chan%, start, finish, &grad{, &inter{, &corr}});
```

**chan%** A channel holding suitable data in the current time or result view.

**start** The start time for processing in a time view, the start bin in a result view.

**finish** The end time for processing in a time view, the end bin in a result view. Data at the **finish** time, or in the **finish** bin, is included in the calculation.

**grad** This is returned holding the gradient of the best fit line (m).

**inter** Optional, returned holding the intercept of the line with the y axis (c).

**corr** Optional, returned holding correlation coefficient indicating the “goodness of fit” of the line. Values close to 1 or -1 indicate a good fit; values close to 0 indicate a very poor fit. This parameter is often referred to as r in textbooks.

Returns 0 if all was OK, or -1 if there were not at least 2 data points.

The results are in user units, so in a time view with a waveform measured in Volts, the units of the gradient are Volts per second and the units of the intercept are Volts. In a result view, the units are y axis units per x axis unit. They are not y units per bin.

### See also:

ChanFit(), ChanMeasure(), FitLinear(), FitPoly(), More about curve fitting

## FitLinear()

This command fits  $y = \mathbf{a}_0f_0(x) + \mathbf{a}_1f_1(x) + \mathbf{a}_2f_2(x) \dots$  to a set of (x,y) data points provided that the functions  $f_n(x)$  are linearly independent. If you can provide error estimates for each y value, you can use the covariance output from this command to provide confidence limits on the calculated coefficients and you can use the returned chi-square value to test if the model is likely to fit the data. The command is:

```
func FitLinear(coef[], y[], x[][]{, s{, covar[][]{, r[]{, mR}}});
```

**coef[]** A real array which sets the number of coefficients to fit and which return the best fit set of coefficients. The array must be between 2 and 10 elements long. The coefficient  $\mathbf{a}_0$  is returned in **coef[0]**,  $\mathbf{a}_1$  in **coef[1]** and so on.

**y[]** A real array of y values.

**x[][]** This array specifies the values of the functions  $f(x)$  at each data point. If there are  $n_c$  coefficients and  $n_d$  data values, this array must be of size at least  $[n_d][n_c]$ . Viewed as a rectangular grid with the coefficients running from left to right and the data running from top to bottom, the values you must fill in are:

$$\begin{array}{cccccc} f_0(x_0) & f_1(x_0) & f_2(x_0) & f_3(x_0) & \dots & f_{n_c-1}(x_0) \\ f_0(x_1) & f_1(x_1) & f_2(x_1) & f_3(x_1) & \dots & f_{n_c-1}(x_1) \\ f_0(x_2) & f_1(x_2) & f_2(x_2) & f_3(x_2) & \dots & f_{n_c-1}(x_2) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ f_0(x_{n_d-1}) & f_1(x_{n_d-1}) & f_2(x_{n_d-1}) & f_3(x_{n_d-1}) & \dots & f_{n_c-1}(x_{n_d-1}) \end{array}$$

1)

`s` This is either a real array holding the standard deviations of the `y[]` data points, or a real value holding the standard deviation of all data points. If `s` is omitted or zero, a least-squares error fit is performed, otherwise a chi-squared fit is done.

`covar` An optional two dimensional array of size at least `[nc][nc]` (`nc` is the number of coefficients fitted) that is returned holding the covariance matrix.

`r[]` An optional array of size at least `[nc]` (`nc` is the number of coefficients fitted) that is returned holding diagnostic information about the fit. The less relevant a fitting function  $f(x)$  is to the fit, the smaller the value returned. The element of the array that corresponds to the most relevant function is returned as 1.0, smaller numbers indicate less relevance.

If your fitting functions are not independent of each other, several coefficients may have low `r` values. The solution is to remove one of the functions from the fit, or to set the `mR` argument to exclude one of the functions, then fit again. If the remaining coefficients become relevant, you have excluded a function that was a linear combination of the others. If the remaining coefficients still are not relevant, you have eliminated a function that did not contribute to the fit.

`mR` You can use this optional variable to set the minimum relevance for a function. Functions that have less relevance than this are “edited” out of the fit and their coefficient is returned as 0. If you do not provide this value, the minimum is set to  $10^{-15}$ , which will probably not exclude any values.

Returns The function returns the chi-square value for the fit if `s[]` or `s` is given (and non-zero), or the sum of squares of the errors between the data points and the best fit line if `s` is omitted or is zero.

The smallest of the sizes of the `y[]` array (and `s[]` array, if provided) and the second dimension of `x[][]` sets the number of data points. It is a fatal error for the number of data points to be less than the number of coefficients.

## An example

This demonstrates how to fit data to the function  $y = a \cdot \sin(x/10) + b \cdot \cos(x/20)$ . The `x` values vary from 0 to 49 in steps of 1. The function `MakeFunc()` calculates a trial data set plus noise. We set `s` to 1.0, so `FitLinear()` returns the sum of squares of the errors between the fitted and input data. If you run this example, you will notice that the returned value is slightly less than the sum of squares of the added errors.

```
const NCOEF%:=2,NDATA%:=50;' coefficient and data sizes
var x[NDATA%][NCOEF%]; ' array of function information
const noise := 0.01; ' controls how much noise we add
var data[NDATA%], err := 0;' space for our function and errors

' Generate raw data. Fit y = a*sin(x/10)+b*cos(x/20)
var coef[NCOEF%], i%, r; ' coefficients, index, random noise
coef[0]:=1.0; coef[1]:=2; ' set coefficients for generated data
MakeFunc(data[], coef[], x[][]); ' Generate the data, then...
for i%:=0 to NDATA%-1 do ' ..add noise to it
 r := (rand()-0.5)*noise;' the noise to add
 data[i%] += r; ' add noise to the data
 err := err + r*r; ' accumulate sum of squared noise
next;

var covar[NCOEF%][NCOEF%]; ' covariance array
var sig2, a[NCOEF%]; ' sigma, fitted coefficients
var rel[NCOEF%]; ' array for "relevance" values
sig2 := FitLinear(a[], data[], x[][] , 1, covar[[]], rel[]);
Message("sig^2=%g, err=%g\ncoefs=%g\nrel=%g",sig2,err,a[],rel[]);
halt;

'y is the output array (x values are 0, 1, 2...), a is the array
'of coefficients. Y = a*sin(x/10)+b*cos(x/20)
proc MakeFunc(y[], a[], x[][])
var nd%,v; ' coefficient index, work space
for nd% := 0 to NDATA%-1 do
 v := Sin(nd% / 10.0); ' first function
 x[nd%][0] := v; ' save the value;
 y[nd%] := a[0] * v; ' start to build the result
 v := Cos(nd%/20.0); ' second function
 x[nd%][1] := v; ' save it
 y[nd%] += a[1]*v; ' full result
next;
```

```
end;
```

**See also:**

More about curve fitting, `ChanFit()`, `FitExp()`, `FitGauss()`, `FitNLUser()`, `FitPoly()`, `FitSin()`

## FitNLUser()

This command uses a non-linear fitting algorithm to fit a user-defined function to a set of data points. The function to be fitted is of the form  $y = f(x, \mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \dots)$  where the  $\mathbf{a}_i$  are coefficients to determine. You must be able to calculate the differential of the function  $f$  with respect to each of the coefficients. You can optionally supply an array to weight each data point. The commands to implement this are:

### Set up the problem

The first command sets the user-defined function, the number of coefficients you want to fit, the number of data points and optionally, you can set the weight to give each data point. You must call this function before you call any of the others.

```
func FitNLUser(User(ind%, a[], dyda[]), nCoef%, nData%{, s[]|s});
```

`User()` A user-defined function which is called by the fitting routine. The function is passed the current values of the coefficients. It returns the error between the function and the data point identified by `ind%` and the differentials of the function with respect to each of the coefficients at that point. The return value should be the y data value at the index minus the calculated value of the function at the x value, using the coefficients passed in.

`ind%` The index into the data points at which to evaluate the error and differentials. If there are  $n$  data points, `ind%` runs from 0 to  $n-1$ . You can rely on the function being called with the same coefficients as `ind%` increments from 0 to  $n-1$ , which may be useful if you have complex functions of the coefficients to evaluate.

`a` An array of length `nCoef%` holding the current values of the coefficients. The coefficients are refreshed for each call to the user-defined function, so it is not an error to change them; however this is usually not done.

`dyda` An array of length `nCoef%` which your function should fill in with the values of the partial differential of the function with respect to each of the coefficients. For example, if you were fitting  $y = \mathbf{a}_0 * \exp(-\mathbf{a}_1 * x)$  then set `dyda[0] =  $\delta y / \delta \mathbf{a}_0 = \exp(-\mathbf{a}_1 * x)$`  and `dyda[1] =  $\delta y / \delta \mathbf{a}_1 = -\mathbf{a}_0 * \mathbf{a}_1 * \exp(-\mathbf{a}_1 * x)$` .

`nCoef%` The number of coefficients to fit in the range 1 to 10.

`nData%` The number of data points you will be fitting. If `s[]` is provided as an array, the value of `nData%` used is the smaller of `nData%` and the length of the `s[]` array. It is a fatal error for the number of data points used to be less than `nCoef%`.

`s` This argument is optional. It is either an array of weights to be given to each data point in the fit or a single weight to apply to all data points. If this value is the expected standard deviation of the y value of the data points, then the error value returned is the chi-squared value and the fit is a chi-squared fit. If this value is proportional to the expected error at the data point, then the fit is still a chi-squared fit, and the error returned is proportional to the chi-squared value. If you omit this argument, the fit is a least-squares error fit, and the error value returned is the sum of the squares of the errors in the y values.

Returns The function returns 0. There is no other return value as all errors stop the script.

Unlike the other fitting routines, you will notice that the x and y data values are not passed into the command. Instead, the user-defined function is passed an index to the data values. It is assumed that the data is accessible by the user function.

Due to restrictions in the implementation of the script language, you cannot debug through the user-defined function. If you set a break point in it, or attempt to step into it you will get errors. We recommend that you check the returned values from the user-defined function by calling it from your own script code.



## Set coefficient values and ranges

This variant of the function sets the initial value of each coefficient and optionally sets the range of allowed values. You can call this function at any time after the setup call and before the iterate call (below) has returned 0 indicating that the fit is completed.

In this type of fitting it is very important that you give reasonable starting values for the coefficients. If you do not give starting values, the command will set them all to zero, which is unlikely to be correct.

```
func FitNLUser(coef%, val{, lo, hi});
```

`coef%` The coefficient to set. The first coefficient is number 0, the last is `nCoef%-1`.

`val` The initial value to assign to the coefficient. If you have set low and high limits, and the value is outside these limits, it is set to the nearer limit.

`lo,hi` If present, these two values set the acceptable range of values for this coefficient. If omitted, or if both values are set to the same value, there is no limit. The value of the coefficient is tested against these limits after each iterative step, and if it exceeds a limit, it is set to that limit.

## Iterate to a solution

Once you have set up the problem and given initial values to your coefficients, you can start the iteration process that will move the coefficients from their starting values to new values that minimise the error (optionally scaled by the `s` argument).

```
func FitNLUser(a[], &err{, maxI{, &iTer{, covar[[[]]]});
```

`a[]` An array of size at least `nCoef%` that is returned holding the current set of coefficient values.

`err` A real variable returned as the sum over the data points of  $(yx[i]-y[i])^2/s[i]^2$  if `s[]` is used or holding the sum of  $(yx[i]-y[i])^2$  if `s[]` is not used, where `yx[i]` is the value predicted from the coefficients at the `x` value `x[i]`.

`maxI%` This is the maximum number of fitting iterations to do before returning from the function. If you omit this value, the function sets 1000. You can set any value from 1 to 10000. If you set more than 10000, the number is limited to 10000.

`iTer%` An optional integer variable that is returned holding the number of iterations done before the function returned.

`covar` An optional two dimensional array of size at least `[nCoef%][nCoef%]` that is returned holding the covariance matrix when the fit is complete. It is not changed unless the function return value is 0 or -1.

Returns 1 if the number of iterations are exhausted and the fit has not converged, 0 if the fit has converged, -1 if the fit is not improving but the result may be OK, -2 if the matrix built to solve the problem is singular (indicating that the coefficients are not linearly independent). Other negative numbers indicate failures like out of memory.

Remember that even when a minimum is found, there is no guarantee that it is the minimum. It is the best minimum that this algorithm can find given the starting point.

### *return value -1*

This can mean that you have started the fit at a minimum point in the error function and the system is unable to make more progress. You can sometimes recover by adjusting your initial guess. We have seen this when we start by fitting a model with `n` free parameters, then change to one with `n+1` and the added parameter is similar in effect to one of the previous ones and we have started it at the same value as the previous one. The previous solution was at a minimum in the error space, and it happened that setting the two parameters the same ended up at the same minimum.

### *return value -2*

The fitting process uses the user defined function to build a matrix, then inverts the matrix to solve the problem. It is possible for this matrix to be singular (which means that a factor used for division has become so small that the result has lost all accuracy). This is usually caused by the coefficients not being mathematically independent. If any coefficient can be derived from the others, it is not independent. If you have a non-independent coefficient, you must recast the equations to fix this. You can also get this situation if in some range of the calculation, a pair of coefficients become related (due to other terms becoming very small, for

instance). You may be able to work around this by restarting with a different guess, or by iterating through the coefficients, holding one constant at a time.

### Select coefficients to fit

Normally the command fits all the coefficients, but you can use this variant to select the coefficients to fit. You can use this command at any time after you have set the problem until the iteration variant returns 0 or -1.

```
func FitNLUser(fit%[]);
```

`fit%[]` An array of at least `nCoef%` integers. If `fit%[i]` is 0, coefficient `i` is held constant, otherwise it is fitted. If all elements are 0, all arguments are fitted.

The effect of this command persists until either the iteration variant returns a value less than 1, or you set up a new problem, or you call this variant again.

#### See also:

Simple example, More complex example, More about curve fitting, `FitExp()`, `FitGauss()`, `FitLinear()`, `FitPoly()`, `FitSin()`

## A simple example

The following is an example of using this set of commands to fit the user-defined function  $y = a * \exp(-b*x)$ . In this example we generate some test data and add to it a random error. There are two coefficients to be fitted (`a` and `b`). To use `FitNLUser()` we need to calculate the differential of the function with respect to `a` and `b`:

```
dy/da = exp(-b*x)
dy/db = -x * a * exp(-b*x)
```

These are standard results. Differential calculus is too large a subject to attempt a summary in this help. If you need to refresh your knowledge, this reference may help.

```
const NDATA%:=100, NCOEF% := 2;
var x[NDATA%],y[NDATA%],i%;
for i% := 0 to NDATA%-1 do ' Generate data
 x[i%] := i%; ' a:=1, b:=0.05 and add some noise
 y[i%] := exp(-0.05*i%)+(rand()-0.5)*0.01;
next;

FitNLUser(UserFnc, NCOEF%, NDATA%); 'Link user function
FitNLUser(0, 0.5, 0.01, 2); 'Set range of amplitude
FitNLUser(1, 0.01, 0.001, 1); 'Set range of exponent

var coefs[NCOEF%], err, iter%;
i% := FitNLUser(coefs[], err, 100, iter%);
Message("fit=%d, Err=%g, iter=%d, coefs=%g",i%,err,iter%,coefs[]);
halt;

' The user-defined function: y = a * exp(-b*x);
' dy/da = exp(-b*x), dy/db = -x * a * exp(-b*x)
func UserFnc(ind%, a[], dyda[])
var xi,yi,r;
xi := x[ind%]; ' local copy of x value
yi := y[ind%]; ' local copy of y value
dyda[0] := exp(-a[1]*xi); ' differential of y with respect to a
r := dyda[0] * a[0]; ' intermediate value
dyda[1] := -xi * r; ' differential of y with respect to b
return yi-r;
end
```

The script is in 4 sections:

1. The header declares the number of data points and the number of coefficients plus space to store a set of `x,y` data points. It then fills in the data points with the function  $y = \exp(-0.05*x)$  and adds some random numbers in the range -0.005 up to 0.005. This will be the test data for the fitting.
2. The second section sets up the problem by telling `FitNLUser()` the name of the user function, the number of data points and the number of coefficients. Then for each coefficient, it sets a starting guess value and a range of acceptable values. The better your initial guess is, the more likely you are to get a useful result.

- Sometimes you will know a likely range of values for each coefficient, on other occasions, you may be able to make some reasonable guess from the initial data. Conversely, if you make a perversely bad initial guess, you may get results that are totally useless. It is not a good idea to set limits that are too close to the guess; sometimes a function has to chase a minimum along a tortuous route and setting very tight limits may prevent it doing this.
3. The third section declares space for the coefficients, tells the function to iterate a maximum of 100 times and prints the results.
  4. The fourth section holds the user function. This is called once for each data point per iteration, so try to minimise the calculations required. Within each iteration, it is called with the index argument having the values 0 to the number of data points minus 1. For each data point we must fill in the differential of the fitting function with respect to each coefficient and we must return the error between the y value and the fitting function.

## Fitting a Gabor function

This demonstrates the use of the `FitNLUser()` script function. This is used to fit to a one dimensional Gabor function (the product of a Gaussian and a cosine wave) defined as:

$$y(x) = b + a \cdot \exp(-g \cdot x \cdot x \cdot 0.5) \cdot \sin(l \cdot x + p)$$

where  $b$  is an offset,  $a$  is the amplitude,  $g$  is twice the variance of the Gaussian,  $l$  is  $2 \cdot \pi / \text{wavelength}$  of the sinusoid and  $p$  is the phase of the sinusoid. There are 5 parameters to fit, being:  $b$ ,  $a$ ,  $g$ ,  $l$ ,  $p$ . The script starts with a header that defines the number of data points and number of coefficients, plus how much noise to add (to make this a more realistic example). It also sets the values of the coefficients that we are to attempt to fit.

```

'$FitGABOR|Demonstration of using the NL fitting to fit a Gabor function
'Author: Greg Smith, Cambridge Electronic Design Limited
const NDATA%:=100, NCOEF% := 5;
var x[NDATA%],y[NDATA%],i%, xv;
' coef# 0 1 2 3 4
const b:=0.002, a:=1.00, g:=0.01, l:=0.1, p:=0.0;
const noise := 0.04; ' maximum noise amplitude to add

' Phase 1. Generate some sample data at equal-spaced x values. If you want
' other than x increments of 1, set xv to i%*width where width is
' the desired x spacing. Note that random noise is added to the
' data values.
for i% := 0 to NDATA%-1 do ' Generate data
 xv := i%;
 x[i%] := i%; ' a:=1, b:=0.05 and add some noise
 y[i%] := b + a*exp(-g*xv*xv*0.5)*Sin(l*xv+p)+(rand()-0.5)*noise;
next;

' Phase 2. Set up the problem. This tells the script language the name of
' the function to use to get the data values, then sets a range of
' acceptable values for each argument and a starting guess for each
' value. It is MOST important that the starting guess is reasonable.
' If it is not, the fitting may yield stupid results.
FitNLUser(UserFnc, NCOEF%, NDATA%); 'Link user function
FitNLUser(0, 0, -0.1, 0.1); 'Set range of offset b
FitNLUser(1, 0.3, 0.5, 2); 'Set range of amplitude a
FitNLUser(2, 0.04, 0.01, 0.1);'Set range of g (1/sigma squared)
FitNLUser(3, 0.03, 0.01, 0.3);'Set range of omega (frequency) l
FitNLUser(4, 1, -3, 3); 'Set range of the phase p

' Phase 3. Solve the problem. This tells the system to improve the initial
' guess. Each iteration will call the User function NDATA% times.
' The result is 0 if it worked, err is the sum of squares of the
' difference between the fitted curve and the raw data, iterations
' is the number of times round the function went and coefs are the
' fitted parameters.
var coefs[NCOEF%], err, iter%, covar[NCOEF%][NCOEF%];
i% := FitNLUser(coefs[], err, 100, iter%, covar[][]);

Message("fit=%d, Err=%.4f, iterations=%d, coefs=%.4f", i%, err ,iter% ,coefs[]);

' If you need to know the likely errors in the fitted parameters, this is

```

```
' available from the covar array (co-variance). The diagonal of the array
' holds the expected variance (sigma squared) of each parameter, on the
' assumption that the errors in the original data have a normal distribution.
'
' See the on-line documentation for the FitNLSUser() script function and read the
' linked section "More about curve fitting" for additional information.
halt;

' The next section is the user-defined function. Differentials of functions with
' several factors will have common expressions that need only be calculated once.
' Although the functions is quite complicated, the calculations turn out to be
' relatively simple.
'
' The user-defined function: y = b + a*exp(-g*x*x*0.5)*Sin(l*x+p);
' dy/db = 1;
' dy/da = exp(-g*x*x*0.5)*Sin(l*x+p)
' dy/dg = -x*x*0.5*a*exp(-g*x*x*0.5)*Sin(l*x+p)
' dy/dl = a*x*exp(-g*x*x*0.5)*Cos(l*x+p)
' dy/dp = a*exp(-g*x*x*0.5)*Cos(l*x+p)

func UserFnc(ind%, a[], dyda[])
var yv,xi,yi,r, xsq2, gauss, angle;
xi := x[ind%]; ' local copy of x value
yi := y[ind%]; ' local copy of y value
xsq2 := xi*xi*0.5; ' calculate once as used twice
gauss := Exp(-a[2]*xsq2); ' calculate once as used twice
angle := a[3]*xi+a[4]; ' used by Sin() and Cos()

dyda[0] := 1; ' dy/db is 1
dyda[1] := gauss*Sin(angle); ' exp(-g*x*x*0.5)*Sin(l*x+p)
dyda[2] := -xsq2*a[1]*dyda[1]; ' -x*x*0.5*a*exp(-g*x*x*0.5)*Sin(l*x+p)
dyda[4] := a[1]*gauss*Cos(angle); ' a*exp(-g*x*x*0.5)*Cos(l*x+p)
dyda[3] := dyda[4]*xi; ' a*x*exp(-g*x*x*0.5)*Cos(l*x+p)
yv := a[0] + a[1]*dyda[1];
return yi-yv;
end
```

## FitPoly()

This command fits  $y = a_0 + a_1x + a_2x^2 + a_3x^3 \dots$  to a set of (x,y) data points. If you can provide error estimates for each y value, you can use the covariance output from this command to provide confidence limits on the calculated coefficients and you can use the returned chi-squared value to test if the model is likely to fit the data. The command is:

```
func FitPoly(coef[], y[], x[]{, s[]|s{, covar[][]});
```

- coef[]** A real array that sets the number of coefficients and returns the coefficient values. The array must be between 2 and 10 elements long. The coefficient  $a_0$  is returned in `coef[0]`,  $a_1$  in `coef[1]` and so on.
- y[]** A real array of y values. The smaller of the sizes of the `x[]` and `y[]` arrays (and `s[]` array, if provided), sets the number of data points. It is a fatal error for the number of data points to be less than the number of coefficients.
- x[]** A real array of x values.
- s** This is an optional argument. It is either a real array holding the standard deviations of each of the `y[]` data points, or it is a real value holding the standard deviation of all of the data points. If the argument is omitted or set to zero, a least-squares error fit is performed, otherwise a chi-squared fit is done.
- covar** An optional two dimensional array of size at least `[nc][nc]` (`nc` is the number of coefficients fitted) that is returned holding the covariance matrix.

**Returns** The function returns chi-squared if `s[]` or `s` is given (and non-zero), otherwise it returns the sum of squares of the errors between the raw and fitted data.

### An example

This example generates test data, adds random noise, then fits a polynomial to the data.

```

const NCOEF% := 5; ' number of coefficients
const NDATA%:=50; ' number of data points
var y[NDATA%], x[NDATA%]; ' space for x and y values for fit
const noise := 1; ' noise to add
var err := 0.0; ' will be sum of squares of added noise
var cf[NCOEF%], i%, r;
cf[0]:=1.0; cf[1]:=-80; cf[2]:=-2.0; cf[3]:=0.5; cf[4]:=-0.009;
MakePoly(cf[],x[],y[]); ' generate ideal data as polynomial
for i%:=0 to NDATA%-1 do ' now add some noise to it
 r := (rand()-0.5)*noise;
 y[i%] += r; ' add noise to the data
 err += r*r; ' sum of squares of added noise
next;
var sig2, a[NCOEF%]; ' a[] will be the fitted coefficients
sig2 := FitPoly(a[], y[], x[]);
Message("sig2=%g, noise=%g\nfitted=%8.4f\nideal =%8.4f",
 sig2, err, a[], cf[]);

halt;
'a[] input array of coefficients
'x[] output x co-ordinates, y[] output data values
proc MakePoly(a[], x[], y[])
var i%,j%,xv,s;
for i% := 0 to Len(y[])-1 do
 s := 0.0;
 xv := 1;
 for j% := 0 to NCOEF%-1 do
 s += a[j%]*xv;
 xv *= i%;
 next;
 y[i%] := s;
 x[i%] := i%;
next;
end;

```

**See also:**

More about curve fitting, `ChanFit()`, `FitExp()`, `FitGauss()`, `FitLinear()`, `FitNLUser()`, `FitSigmoid()`, `FitSin()`

## FitSigmoid()

This command a single Sigmoid function to x,y data points with an optional weight for each point. Fitting is by an iterative method. The input data is fitted to the equation:

$$y = \mathbf{a}_0 + (\mathbf{a}_1 - \mathbf{a}_0) / (1 + \exp((\mathbf{a}_2 - x) / \mathbf{a}_3))$$

The fitted parameters (coefficients) are the  $\mathbf{a}_i$ . You can only fit 1 Sigmoid .In terms of the fitted result,  $\mathbf{a}_0$  and  $\mathbf{a}_1$  are the low and high fitting limits,  $\mathbf{a}_2$  is the X50 point and  $\mathbf{a}_3$  is related to the reciprocal of the slope at the X50 point. The slope at the X50 point, where x is  $\mathbf{a}_2$ , is  $(\mathbf{a}_1 - \mathbf{a}_0)/(4 \mathbf{a}_3)$ . The commands to implement this are:

### Set up the problem

The first command sets the number of Sigmoids to fit and the data set to be fitted. You must call this function before you call any of the others.

```
func FitSigmoid(nCoef%, y[], x[][, s[]|s]);
```

**nCoef%** The number of coefficients to fit. The only legal value is 4 for one Sigmoid.

**y[]** An array of y data values. The length of the array must be at least nCoef%.

**x[]** An array of x data values. The length of the array must be at least nCoef%.

**s** An optional argument which is either an array with one value for each y data point or a single value for all data points. If the value is the standard deviation, then the error value returned when you iterate to find the best fit is the chi-squared value and the fit is a chi-squared fit.

If this value is proportional to the error in the y values, then the fit is still a chi-squared fit, and the error returned is proportional to the chi-squared value. If you omit this array, the fit is a least-squares error fit, and the error value returned is the sum of the squares of the errors in the y values.

Returns The function returns 0. There is no other return value as all errors stop the script.

The number of data points is set by the smallest of the sizes of the `y[]`, `x[]` or `s[]` (if present) arrays. The number of data points must be at least the number of coefficients. If it is not, you will get a fatal error, so check this before calling the function.

### Set coefficient values and ranges

This variant of the function sets the initial value of each coefficient and optionally sets the range of allowed values. You can call this function at any time after the setup call and before the iterate call (below) has returned 0 indicating that the fit is completed.

In this type of fitting it is very important that you give reasonable starting values for the coefficients. In particular, when fitting a Sigmoid, it is usual that the two levels (`a0` and `a1`) are easy to determine. If you can set the levels and limit them so that they cannot overlap, the fit usually will proceed without any problems. If you do not give starting values, the command will make a simplistic guess at the fitting values. As we expect that you know more about the “right” answer than the command does, we suggest that you set the values you want.

```
func FitSigmoid(coef%, val{, lo, hi});
```

`coef%` The coefficient to set. The first coefficient is number 0, the last is `nCoef%-1`.

`val` The initial value to assign to the coefficient. If you have set low and high limits, and the value is outside these limits, it is set to the nearer limit.

`lo,hi` If present, these two values set the acceptable range of values for this coefficient. If omitted, or if both values are set to the same value, there is no limit. The value of the coefficient is tested against these limits after each iterative step, and if it exceeds a limit, it is set to that limit.

As long as you make a reasonable estimate of the two levels, there should be no problems fitting a Sigmoid.

### Iterate to a solution

Once you have set up the problem and given initial values to your coefficients, you can start the iteration process that will move the coefficients from their starting values to new values that minimise the error (optionally scaled by the `s` argument).

```
func FitSigmoid(a[], &err{, maxI{, &iTer{, covar[][]});
```

`a[]` An array of size at least `nCoef%` that is returned holding the current set of coefficient values. The first amplitude is in `a[0]`, the first centre in `a[1]`, the first sigma in `a[2]`, the second amplitude in `a[3]` and so on.

`err` A real variable returned as the sum over the data points of  $(yx[i]-y[i])^2/s[i]^2$  if `s[]` is used or holding the sum of  $(yx[i]-y[i])^2$  if `s[]` is not used, where `yx[i]` is the value predicted from the coefficients at the `x` value `x[i]`.

`maxI%` This is the maximum number of fitting iterations to do before returning from the function. If you omit this value, the function sets 1000. You can set any value from 1 to 10000. If you set more than 10000, the number is limited to 10000.

`iTer%` An optional integer variable that is returned holding the number of iterations done before the function returned.

`covar` An optional two dimensional array of size at least `[nCoef%][nCoef%]` that is returned holding the covariance matrix when the fit is complete. It is not changed unless the function return value is 0 or -1.

Returns 1 if the number of iterations are exhausted and the fit has not converged, 0 if the fit has converged, -1 if the fit is not improving but the result may be OK. Other negative numbers indicate failure.

Remember that even when a minimum is found, there is no guarantee that this is the minimum. It is the best minimum that this algorithm can find given the starting point.

### Select coefficients to fit

Normally the command fits all the coefficients, but you can use this variant to select the coefficients to fit. You can use this command at any time after you have set the problem until the iteration variant returns 0 or -1.

```
func FitSigmoid(fit%[]);
```

`fit%[]` An array of at least `nCoef%` integers. If `fit%[i]` is 0, coefficient `i` is held constant, otherwise it is fitted. If all elements are 0, all arguments are fitted.

The effect of this command persists until either the iteration variant returns a value less than 1, or you set up a new problem, or you call this variant again.

## An example

This is a template for using these commands to fit all the coefficients:

```
const nData%:=50; 'set number of data elements
var x[nData%], y[nData%]; 'space for our arrays
var s[nData%]; 'space for sigma of each point
var coefs[4], err; 'coefficients and error squared
... 'in here goes code to get the data
FitSigmoid(4, y[], x[], s[]); 'fit one gaussian
FitSigmoid(0, 1.0, 0.2, 4); 'set base level and limit range
FitSigmoid(1, 20, 15, 25); 'set end level and range
FitSigmoid(2, 8); 'initial 50% point in X units
FitSigmoid(3, 0.5); 'initial slope
repeat
 DrawMyData(coefs[], x[], y[]); 'Some function to show progress
until FitSigmoid(coefs[], err, 1) < 1;
DrawMyData(coefs[], x[], y[]); 'Show the final state
```

## See also:

More about curve fitting, `ChanFit()`, `FitExp()`, `FitLinear()`, `FitGauss()`, `FitNLUser()`, `FitPoly()`, `FitSin()`

## FitSin()

This command fits multiple sinusoids to arrays of x,y data points, with an optional weight for each point. Fitting is by an iterative method. The input data is fitted to the equation:

$$y = \mathbf{a}_0 \sin(\mathbf{a}_1 x + \mathbf{a}_2) + \mathbf{a}_3 \sin(\mathbf{a}_4 x + \mathbf{a}_5) + \dots \{+ \mathbf{a}_{3n}\}$$

The fitted coefficients are the  $\mathbf{a}_i$ . Angles are evaluated in radians. You can fit up to 3 sinusoids and an optional offset. Although the function is given in terms of sine functions, you can easily convert to cosines by subtracting  $\pi/2$  from the phase angle ( $\mathbf{a}_2, \mathbf{a}_5, \mathbf{a}_8$ ) after the fit. The commands to implement this are:

## Set up the problem

The first command sets the number of sinusoids to fit and the data set to be fitted. You must call this function before you call any of the others.

```
func FitSin(nCoef%, y[], x[][, s[]|s]);
```

`nCoef%` The number of coefficients to fit. The only legal values are 3, 6 and 9 for one, two and three sinusoids or 4, 7 and 10 to include an offset as the last coefficient.

`y[]` An array of y data values. The length of the array must be at least `nCoef%`.

`x[]` An array of x data values. The length of the array must be at least `nCoef%`.

`s` An optional argument which is either an array with one value for each y data point or a single value for all data points. If the value is the standard deviation, then the error value returned when you iterate to find the best fit is the chi-squared value and the fit is a chi-squared fit.

If this value is proportional to the error in the y values, then the fit is still a chi-squared fit, and the error returned is proportional to the chi-squared value. If you omit this array, the fit is a least-squares error fit, and the error value returned is the sum of the squares of the errors in the y values.

Returns The function returns 0. There is no other return value as all errors stop the script.

The number of data points is set by the smallest of the sizes of the `y[]`, `x[]` or `s[]` (if present) arrays. The

number of data points must be at least the number of coefficients. If it is not, you will get a fatal error, so check this before calling the function.

### Set coefficient values and ranges

This variant of the function sets the initial value of each coefficient and optionally sets the range of allowed values. You can call this function at any time after the setup call and before the iterate call (below) has returned 0, indicating that the fit is complete.

In this type of fitting it is very important that you give reasonable starting values for the coefficients. In particular, when fitting multiple sinusoids you will usually either know, or have a good idea of the frequencies. You should limit the range of each frequency so that they cannot overlap. If you can do this, the fit will proceed quickly. If you do not give starting values, the command will make a simplistic guess at the fitting values. As we expect that you know more about the "right" answer than the command does, we suggest that you set the values you want.

```
func FitSin(coef%, val{, lo, hi});
```

`coef%` The coefficient to set. The first coefficient is number 0, the last is `nCoef%-1`.

`val` The initial value to assign to the coefficient. If you have set low and high limits, and the value is outside these limits, it is set to the nearer limit.

`lo,hi` If present, these two values set the acceptable range of values for this coefficient. If omitted, or if both values are set to the same value, there is no limit. The value of the coefficient is tested against these limits after each iterative step, and if it exceeds a limit, it is set to that limit.

### Iterate to a solution

Once you have set up the problem and given initial values to your coefficients, you can start the iteration process that will move the coefficients from their starting values to new values that minimise the error (optionally scaled by the `s` argument).

```
func FitSin(a[], &err{, maxI%{, &iTer%{, covar[][]}}});
```

`a[]` An array of size at least `nCoef%` that is returned holding the current set of coefficient values. The first amplitude is in `a[0]`, the first frequency in `a[1]`, the first phase angle in `a[2]`, the second amplitude in `a[3]` and so on.

`err` A real variable returned as the sum over the data points of  $(yx[i]-y[i])^2/s[i]^2$  if `s[]` is used or holding the sum of  $(yx[i]-y[i])^2$  if `s[]` is not used, where `yx[i]` is the value predicted from the coefficients at the `x` value `x[i]`.

`maxI%` This is the maximum number of fitting iterations to do before returning from the function. If you omit this value, the function sets 1000. You can set any value from 1 to 10000. If you set more than 10000, the number is limited to 10000.

`iTer%` An optional integer variable that is returned holding the number of iterations done before the function returned.

`covar` An optional two dimensional array of size at least `[nCoef%][nCoef%]` that is returned holding the covariance matrix when the fit is complete. It is not changed unless the function return value is 0 or -1.

Returns 1 if the number of iterations are exhausted and the fit has not converged, 0 if the fit has converged, -1 if the fit is not improving but the result may be OK. Other negative numbers indicate failure.

Even when a minimum is found, there is no guarantee that this is the minimum, only that it is the best minimum that this algorithm can find given the starting point.

### Select coefficients to fit

Sometimes you may wish to hold some coefficients fixed while you fit others. Normally the command will fit all the coefficients, but you can use this command variant to select the coefficients to fit. You can use this command at any time after you have set the problem until the iteration variant returns 0 or -1.

```
func FitSin(fit%[]);
```

`fit%[]` An array of at least `nCoef%` integers. If `fit%[i]` is 0, coefficient `i` is held constant, otherwise it is



fitted. If all elements are 0, all arguments are fitted.

The effect of this command persists until either the iteration variant returns a value less than 1, or you set up a new problem, or you call this variant again. For a sinusoidal fit it is likely that you will know the frequency to fit, so you may well hold this constant.

### An example

The following is a template for using this command (assuming you don't want to fit the frequency, which we assume you know).

```
const nData:=50; 'set number of data elements
var x[nData:], y[nData:]; 'space for our arrays
var s[nData:]; 'space for sigma of each point
var fit%[3]; 'we want to hold the frequency
var coefs[4]; 'space for coefficients
var err; 'will hold error squared
... 'in here goes code to get the data
FitSin(3, y[], x[], s[]); 'fit one sinusoid
'Note that we let the phase take any value
FitSin(0, 1.0, 0.2, 4); 'set amplitude and limit range
FitSin(1, .02, .01, .03); 'set frequency
FitSin(2, 0., 0.3, 1.9); 'set width and limit range
'Now we say that we don't want to fit the frequency
ArrConst(fit%[],1); 'set all elements to 1
fit%[1] := 0; 'but not element 1 (=frequency)
FitSin(fit%[]); 'so the frequency is fixed
repeat
 DrawMyData(coefs[], x[], y[]); 'Some function to show progress
until FitSin(coefs[], err, 1) < 1;
DrawMyData(coefs[], x[], y[]); 'Show the final state
```

### See also:

More about curve fitting, `FitExp()`, `FitGauss()`, `FitLinear()`, `FitNLUser()`, `FitPoly()`, `FitSigmoid()`

## FitValue()

This function returns the value at a particular x position of the fitted function set by the last `FitData()` command.

```
Func FitValue(x{, &valid%});
```

**x** The x value at which to evaluate the current fit. You should be aware that some of the fitting functions can overflow the floating point range if you ask for x values beyond the fitted range of the function.

**valid%** If present, this integer variable is set to 1 if the returned value is valid, 0 if not.

**Returns** The value of the fitted function at x. If the result is out of floating point range, the function may return a floating point infinity or a NaN (Not a Number) value or a 0. If there is no fit, the result is always 0.

### See also:

`FitCoef()`, `FitData()`, `FitExp()`, `ChanFitValue()`, More about curve fitting

## Floor()

Returns the next lower integral number of the real number or array. `Floor(4.7)` is 4.0, `Floor(4)` is 4. `Floor(-4.7)` is -5.

```
Func Floor(x|x[]{[]...});
```

**x** A real number or a real array.

**Returns** 0 or a negative error code for an array or the next lower integral value.

**See also:**

Abs(), ATan(), Ceil(), Cos(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan(), Trunc()

## FocusHandle()

This function returns the view handle of the script-controllable window with the input focus (the active window). Unlike `FrontView()`, it can return any type of window, for example multimedia, cursor regions and spike shape windows.

```
Func FocusHandle();
```

Returns The handle of a window that the script can manipulate, or 0 if the focus is not in such a window.

**See also:**

`FrontView()`

## FontGet()

This function gets the name of the font, and its characteristics for the current view.

```
Func FontGet(&name$, &size, &flags%, {, style%{, &fore%{, &back%}}});
```

`name$` This string variable is returned holding the name of the font.

`size` The real number variable is returned holding the point size of the font.

`flags%` Returned holding the sum of the style values: 1=Italic, 2=Bold, 4=Underline, 8=Force upper case, 16=force lower case. Only one of 8 or 16 will be returned.

`style%` Text-based views have all have default style (32) that is used as the basis of all other styles, plus a number of additional styles that are used to highlight keywords and the like in output sequencer and script windows. The style codes for other styles start at 0 and run upwards. You can find a list of styles for each view type in the Edit Preferences General tab in the Editor settings section. If you omit `style%` the settings for the default style (32) are returned.

All other views ignore `style%` (which should be set to 0, if supplied) except time views, which use style 0 for the normal view style and style 1 for the font for vertical markers.

`fore%` This value returns the foreground colour of the style. Bits 0-7 hold the red intensity, bits 8-15 hold the green and bits 16-23 hold the blue.

`back%` This value returns the background colour of the style.

Returns The function returns 0 if all was well or a negative error code. If an error occurs, the variables are not changed.

The arguments from `style%` onwards and the `flag%` values 4, 8 and 16 only apply to text-based views and are new in version 6.

**See also:**

`FontSet()`, `TabSettings()`

## FontSet()

This function sets the font for the current view. Text-based views (text, sequencer and script) normally avoid proportionally spaced fonts as they did not display correctly before Spike2 version 6. Arguments from `style%` onwards are for text-based views.

```
Func FontSet(name$|code%, size, flags%, {, style%{, fore%{, back%}}});
```

`name$` A string holding the font name to use or you can specify a font code:

`code%` This is an alternative method of specifying a font. We recognise these codes:

0 The standard system font, whatever that might be  
 1 A non-proportionally spaced font, usually Courier-like  
 2 A proportionally spaced non-serifed font, such as Helvetica or Arial  
 3 A proportionally spaced serifed font, such as Times Roman  
 4 A symbol font  
 5 A decorative font, such as Zapf-Dingbats or TrueType Wingdings

**size** The point size required. Your system may limit the allowed range.

**flags%** The sum of the style values to set: 1=Italic, 2=Bold, 4=Underline, 8=Force upper case, 16=force lower case. If both 8 and 16 are set, 16 is ignored. Values from 4 upwards are only supported by text-based views.

**style%** Text-based views have a default style (32) plus a number of additional styles that are used to highlight keywords in output sequencer and script windows. The style codes for other styles start at 0 and run upwards. You can find a list of styles for each view type in the Edit Setup dialog. If you omit `style%`, the default style is changed. Set the value -1 to set all styles to the values you give.

All other views ignore `style%` (which should be set to 0, if supplied) except time views, which use style 0 for the normal view style and style 1 for the font for vertical markers.

**fore%** This sets the style foreground colour or is set to -1 to make no change. Bits 0-7 hold the red intensity, bits 8-15 hold the green and bits 16-23 hold the blue. Bits 24-31 are 0. It is convenient to code this as a hexadecimal number, for example:

```
const red%=0x0000ff, green%=0x00ff00, blue%=0xff0000;
const gray%=0x808080, yellow% =0x00ffff;
```

**back%** This sets the background colour of the style in the same format as `fore%`.

**Returns** The function returns 0 if the font change succeeded, or a negative error code.

**See also:**  
 Edit setup dialog, `FontGet()`, `TabSettings()`

## Frac()

Returns the fractional part of a real number or converts a real array to its fractional parts.

```
Func Frac(x|x[]{|...});
```

**x** A real number or an array of real numbers.

**Returns** For arrays, it returns 0 or a negative error code. If `x` is not an array it returns a real number equal to `x-Trunc(x)`. `Frac(4.7)` is 0.7, `Frac(-4.7)` is -0.7.

**See also:**

`Abs()`, `ATan()`, `Cos()`, `Exp()`, `Ln()`, `Log()`, `Max()`, `Min()`, `Pow()`, `Rand()`, `Round()`, `Sin()`, `Sqrt()`, `Tan()`, `Trunc()`

## FrontView()

The front view is the time, result, XY or text-based view that decides which menu appears in the main window. You can use this command to get the front view, or to set it. When a view becomes the front view, it is moved to the front and made the current view. If an invisible or iconised view is made the front view, it is made visible automatically, (equivalent to `WindowVisible(1)`).

You can also use this command to move other windows (for example the multimedia, cursor and spike shape windows) to the top and make them the current view, but such windows are not returned by this command. Use `FocusHandle()` to return these windows.

```
Func FrontView({vh%});
```

**vh%** Either 0 or omitted to return the front view handle, a view handle to be set, or -n, meaning the n<sup>th</sup> duplicate of the time view associated with the current view.

Returns 0 if there are no visible views, -1 if the view handle passed is not a valid view handle, otherwise it returns the view handle of the view that was at the front.

**See also:**

FocusHandle(), View(), WindowVisible()

**G****GammaP() and GammaQ()**

These functions return the incomplete gamma function  $P(a, x)$ . It is defined as:

$$P(a, x) = \frac{\int_0^x e^{-t} t^{a-1} dt}{\Gamma(a)} \quad \text{where} \quad \Gamma(a) = \int_0^\infty t^{a-1} e^{-t} dt$$

This is named `GammaP()` in the script. We also define the complement of this function, `GammaQ(a, x)`, which is  $1.0 - \text{GammaP}(a, x)$ .

```
Func GammaP(a, x);
```

```
Func GammaQ(a, x);
```

a This must be positive, it is a fatal error if it is not.

x This must be positive, it is a fatal error if it is not.

Returns These functions return the incomplete Gamma function and the complement of the incomplete Gamma function.

These functions are not of that much direct interest. However, from them are obtained the error function, the cumulative Poisson probability function and the Chi-squared probability function.

**See also:**

BetaI(), LnGamma(), Distributions based on GammaP()

**Distributions based on GammaP()****Error function**

The error function  $\text{erf}(x) = \text{GammaP}(0.5, x*x)$

This is defined as:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

which sounds uninteresting until you realise that this is the integral of a Normal distribution. As defined, the result is always positive. If you want a version that gives a negative result for negative x, use:

```
Func Erf(x)
var r;
r := GammaP(0.5, x*x);
return x >= 0 ? r : -r;
end;
```

The functions use becomes more obvious when the  $x$  value is expressed in terms of the standard deviation of a Normal distribution. For example, if you want to know what proportion of normally distributed data lies within  $n$  standard deviations of the mean, you can use:

```
Func NormProp(n)
return GammaP(0.5, n*n*0.5);
end;
```

Multiply the result by 100 to get a percentage. The proportion of data that lies outside  $n$  standard deviations of

the mean is:

```
Func NormPropComp(n)
return GammaQ(0.5, n*n*0.5);
end;
```

### Normal distribution cumulative distribution function

To get the proportion of normally distributed data that lies below  $x$  (from minus infinity to  $x$ ) given a mean and a standard deviation, you can use:

```
'x The value to test
'mean The mean value of the Normal distribution
'sd The standard deviation of the distribution
'Returns The proportion of data that is less than or equal to x
Func NormDistCDF(x, mean, sd)
x -= mean;
x /= sd;
return (x>=0) ? 0.5*(1 + GammaP(0.5, x*x*0.5)) : 0.5*GammaQ(0.5, x*x*0.5);
end;
```

The proportion above  $x$  (from  $x$  to plus infinity) is given by the complement of `NormDistCDF()`:

```
Func NormDistCDFComp(x, mean, sd)
x -= mean;
x /= sd;
return (x>=0) ? 0.5*GammaQ(0.5, x*x*0.5) : 0.5*(1 + GammaP(0.5, x*x*0.5));
end;
```

Do not be tempted by:

```
1.0 - NormDistCDF(x, mean, sd);
```

as this severely limits the accuracy when the result is small (that is when `NormDistCDF()` is close to 1.0).

You can also perform the inverse of these function (that is, given a proportion, what is the value) with `ZeroFind()`. If you are attempting to find values that are very close to 1, you would do much better to use the complement function and search for values that are very close to 0. That is, to find the position where `NormDistCDF()` is 0.99999, you should find the position where `NormDistCDFComp()` is 0.00001 as this will give a much more accurate result.

### Cumulative Poisson probability function

The cumulative Poisson probability function relates to a Poisson process of random events and is the probability that, given an expected number of events  $r$  in a given time period, the actual number was greater than or equal to  $n$ . This turns out to be `GammaP(n, r)`. Also, the probability that there are less than  $n$  events is `GammaQ(n, r)`.

### Chi-squared probability function

The Chi-squared probability function is useful where we are fitting a model to data. Given a fitting function that fits the data with  $n$  degrees of freedom (if you have `nData` data points and `nCoef` coefficients you usually have `nData-nCoef` degrees of freedom), and given that the errors in the data points are normally distributed, the probability of a Chi-squared value less than `chisq` is `GammaP(n/2, chisq/2)`. Similarly, the probability of a `chisq` value at least as large as `chisq` is `GammaQ(n/2, chisq/2)`. So, if you know the chi-squared value from a fitting exercise, you can ask "What is the probability of getting this value (or a greater one) given that my model fits the data?" If the probability is very small, it is likely that your model does not fit the data, or your fit has not converged to the correct solution.

**See also:**

`BetaI()`, `LnGamma()`, Binomial Distribution, Student's T Distribution, F-Distribution

## GammaQ()

The complement of `GammaP()`; `GammaQ(a, x)` is `1.0-GammaP(a, x)`.

```
Func GammaQ(a, x);
```

`a` This must be positive, it is a fatal error if it is not.

x This must be positive, it is a fatal error if it is not.

Returns The complement of the incomplete Gamma function.

**See also:**

GammaP()

## Grid()

This function turns the background grid on and off for the current time, result or XY view and returns the state of the grid. The grid is a mesh of lines that follow the big and small ticks on the x and y axes that is drawn on top of the data view background and under the data. Separate control of x and y grid lines was added at version 7.02.

```
Func Grid({on%});
```

on% Optional, sets the grid state. Omit or set -1 for no change, 0 = no grid, 1 = both x and y grid, 2=x grid only, 3=y grid only.

Returns The state of the grid at the time of the call as 0-3 or a negative error code. Changes made by this function do not cause an immediate redraw.

**See also:**

XAxis(), XScroller(), YAxis()

## Gutter()

The gutter is the area on the left of a text-based window where bookmarks and script break points appear. This function returns and optionally sets the gutter visible state. If you set a large font size you may wish to hide the gutter.

```
Func Gutter({show%});
```

show% Optional, sets the gutter state. 0 = hide, 1 = show, -1 or omitted for no change.

Returns The gutter state at the time of the call: 0 = hidden, 1=visible.

**See also:**

ViewLineNumbers()

## H

### HCursor()

This function returns the position of a horizontal cursor, and optionally sets a new position. You can get and set positions of cursors attached to invisible channels or channels that have no y axis. It is also used with spike shape windows that have horizontal cursors to get and set trigger levels and in an offline Edit WaveMark window to read back the last set values.

```
Func HCursor(num% {,where {,chan%}});
```

num% The cursor to use. It is an error to attempt this operation on an unknown cursor.

In a spike shape window, cursor 1 is the low trigger level, 2 is the high trigger, 3 is the low limit (if enabled), 4 is the high limit (if enabled). To set the cursor for trace n (0-3) add 4\*n to the cursor number.

where If this parameter is given it sets the new position of the cursor.

chan% If this parameter is given, it sets the channel number. In XY or spike shape views you should omit this argument as it is ignored.

Returns The function returns the position of the cursor at the time of the call.

### Use in a Spike shape view

You can use code like the following to set a horizontal cursor level in a spike shape window.

```
var sv%;
sv% := SSOpen(0); 'get handle of open spike shape dialog
if sv% > 0 then View(sv%).HCursor(2,1.3) endif;
```

You can use `HCursor()` in an offline Edit WaveMark view to read back (but not to set) the last set cursor levels. These may be the levels used to create the current set of spikes. However, the user was perfectly at liberty to change the levels after the events were created, but before the spike shape window closed, so you cannot rely on these levels.

#### See also:

`HCursorChan()`, `HCursorDelete()`, `HCursorLabel()`, `HCursorLabelPos()`, `HCursorNew()`, `HCursorRenumber()`

## HCursorChan()

This function returns the channel number that a particular cursor is attached to.

```
Func HCursorChan(num%);
```

num% The horizontal cursor number.

Returns It returns the channel number that the cursor is attached to, or 0 if this cursor is not attached to any channel or if the channel number is out of the allowed range. XY views return 1 as the channel number.

#### See also:

`HCursor()`, `HCursorDelete()`, `HCursorLabel()`, `HCursorLabelPos()`, `HCursorNew()`, `HCursorRenumber()`

## HCursorDelete()

This deletes the designated horizontal cursor. Deleting an unknown cursor has no effect.

```
Func HCursorDelete({num%});
```

num% The number of the cursor to delete. If this is omitted, the highest numbered cursor is deleted. Set -1 to delete all horizontal cursors.

Returns The number of the deleted cursor or 0 if no cursor was deleted.

#### See also:

`HCursor()`, `HCursorChan()`, `HCursorLabel()`, `HCursorLabelPos()`, `HCursorNew()`, `HCursorRenumber()`

## HCursorExists()

Use this function to determine if a horizontal cursor exists.

```
Func HCursorExists(num%);
```

num% The cursor number in the range 1-9.

Returns 0 if the cursor does not exist, 1 if it does.

#### See also:

`HCursorNew()`, `CursorExists()`

## HCursorLabel()

This gets and optionally sets the horizontal cursor label style for the view or for a cursor. You can label cursors with a position and/or the cursor number, or with user-defined text. There are two variants: the first sets styles and labels; the second reads back user-defined labels.

```
Func HCursorLabel({style%{, num%{, form$}}});
Func HCursorLabel(&form$, num%);
```

**style%** Set 0=None, 1=Position, 2=Number, 3=Both, 4=User-defined. Omit for no change. Style 4 is used with a format string. Styles 0-3 set the styles of cursors selected by **num%** and the view style for new cursors if **num%** is -1 or omitted. Style 4 is applied to cursors set by **num%**; it does not set the view style.

**num%** 1-9 to select a single cursor or to return the label string. Omit **num%** or set it less than 1 (use -1 in case we allow a cursor 0 in the future) for all cursors and to get and set the style for new cursors.

**form\$** The user-defined label string for style 4. The string has replaceable parameters %p, and %n for position and number. We also allow %w.dp where w and d are numbers that set the field width and decimal places.

Returns A cursor style before any change as 0-4 if a single cursor is selected, or the view cursor style as 0-3. If **style%** is omitted or not 0-3, the current view cursor style is not changed.

**See also:**

HCursor(), HCursorChan(), HCursorDelete(), HCursorLabelPos(), HCursorNew(), HCursorRenumber()

## HCursorLabelPos()

This lets you set and read the position of the cursor label.

```
Func HCursorLabelPos(num% {, pos});
```

**num%** The cursor number. Setting a silly number does nothing and returns -1.

**pos** If present, the command sets the position. The position is a percentage of the distance from the left of the cursor at which to position the value. Out of range values are set to the appropriate limit.

Returns The cursor position before any change was made.

**See also:**

HCursor(), HCursorChan(), HCursorDelete(), HCursorLabel(), HCursorNew(), HCursorRenumber()

## HCursorNew()

This function creates a new horizontal cursor in a time or result view and assigns it to a channel. You can create up to 9 horizontal cursors.

```
Func HCursorNew(chan% {, where});
```

**chan%** A channel for the new cursor. If the channel is hidden, the cursor is not visible. You should use a channel number of 1 for XY views.

**where** An optional argument setting the cursor position. If this is omitted, the cursor is placed in the middle of the y axis or at zero if there is no y axis.

Returns It returns the horizontal cursor number or 0 if all cursors are in use.

**See also:**

HCursor(), HCursorChan(), HCursorDelete(), HCursorLabel(), HCursorLabelPos(), HCursorRenumber()



## HCursorRenumber()

This command rennumbers the cursors from bottom to top. There are no arguments.

```
Func HCursorRenumber();
```

Returns The number of cursors found in the view.

### See also:

HCursor(), HCursorChan(), HCursorDelete(), HCursorLabel(), HCursorLabelPos(), HCursorNew()

## Help()

This command lets you display help information from the standard Spike2 help file or from a help file of your choosing. You can also use `Help(0)` to close any open help file.

```
Func Help(topic%|topic$ {,file$});
```

`topic%` A numeric code for the help topic. These codes are assigned by the help system author. If `topic%` is zero, this closes any open help file and changes the default help file to `file$` or sets it back to the standard help file if `file$` is omitted or empty.

`topic$` A string holding a help topic keyword or phrase to look-up.

`file$` If this is omitted, or the string is empty, the standard Spike2 help file is used. If this holds a filename, this filename is used as the help file. If `topic%` is present and zero, this sets the name of the help file to use for future help requests until the script terminates.

Returns 1 if the help topic was found, 0 if it was not found, -1 if no help file was found. `Help(0, file$)` returns 0 if the new file exists and -1 if it does not.

The Windows SDK has some help-authoring tools, and third-party tools are available if you wish to generate your own .hlp files.

## I

## IIR commands

The `IIRxxxx()` script commands make it easy for you to generate and apply IIR (Infinite Impulse Response) filters to data held in arrays of real numbers. The data values are assumed to be a sampled sequence, spaced at equal intervals. You can create digital filters that are modelled on Butterworth, Bessel, Chebyshev type 1 and Chebyshev type 2 highpass, lowpass, bandstop and bandpass filters. You can also create digital resonators and notch filters. The commands are:

|                                      |                                      |                                      |
|--------------------------------------|--------------------------------------|--------------------------------------|
| <code>IIRBp()</code> Bandpass filter | <code>IIRHp()</code> Highpass filter | <code>IIRNotch()</code> Notch filter |
| <code>IIRBs()</code> Bandstop filter | <code>IIRLp()</code> Lowpass filter  | <code>IIRReson()</code> Resonator    |

The algorithms used to create the filters are based on the `mkfilter` program, written by Tony Fisher of York University. The basic idea is to position the s-plane poles and zeros for a normalised low-pass filter of the desired characteristic and order, then to transform the filter to the desired type. The poles of a filter must lie outside the unit circle for a filter to be stable. The filter commands can return how close the poles are to the unit circle and this value limits the stability of the generated filters; this distance is termed the *stability value* in the command descriptions and experience shows that values less than  $1e-12$  tend to produce noisy or unstable filters.

### Filter expression

The theory of IIR filters is beyond the scope of this manual; a classic reference work is *Theory and Application*

of *Digital Signal Processing* by Rabiner and Gold, published in 1975. The IIR filters generated by these commands can be modelled by:

$$y[n] = \sum_{i=0,N} (A_i * x[n-i]/G) + \sum_{i=1,M} (B_i * y[n-i])$$

where the  $x[n]$  are the sequence of input data values, the  $y[n]$  are the sequence of output values, the  $A_i$  and the  $B_i$  are the filter coefficients (some of which may be zero) and  $G$  is the filter gain. Although  $G$  could be incorporated into the  $A_i$ , for computational reasons we keep it separate. In the filters designed by the `IIRxxxxx()` commands,  $N=M$  and is the order of the filter for lowpass and highpass designs, twice the order for bandpass and bandstop designs and is 2 for resonators and notch filters. The order of the low pass, high pass, band pass and band stop filters determines the sharpness of the filter cut-off: the higher the order, the sharper the cut-off.

## IIR and FIR filters

When compared to FIR filters, IIR filters have advantages:

- They can generate much steeper edges and narrower notches for the same computational effort.
- The filters are causal, which means that the filter output is only affected by current and previous data. If you run a step change through FIR filters you typically see ringing before the step as well as after it.

However, they also have disadvantages:

- IIR filters are prone to stability problems, FIR filters are unconditionally stable. IIR filters generated by these commands will normally be stable unless very narrow features ( $<0.0001$  of the sample rate) are used. Stability gets worse as the filter order increases. You can get an indication of stability after generating a filter.
- They impose a group delay on the data that varies with frequency. This means that they do not preserve the shape of a waveform, in particular, the positions of peaks and troughs will change.

You can remedy the group delay problem by running a filter forwards, then backwards, through the data. However, this makes the filter non-causal, removing one of the advantages of using an IIR filter. The commands allow you to check the impulse, step, frequency and phase response of the filters, and we recommend that you do so before using a generated filter for a critical purpose.

The lowpass, highpass, bandpass and bandstop filters generate digital filters modelled on four types of analogue filter: Butterworth, Bessel, Chebyshev type 1 and Chebyshev type 2. The resulting digital filters are not identical to the analogue filters as the mapping from the analogue to the digital domain distorts the frequency scale. In many cases, this improves the performance of the digital filter over the analogue counterpart.

## Filter types

You can generate notch and resonator filters plus lowpass, highpass, bandpass and bandstop filters modelled on Butterworth, Bessel and Chebyshev analogue filters.

### Butterworth

These have a maximally flat pass band, but pay for this by not having the steepest possible transition between the pass band and the stop band.

### Bessel

An analogue Bessel filter has the property that the group delay is maximally flat, which means that it tends to preserve the shape of a signal passed through it. This leads to filters with a gentle cut-off. When digitised, the constant group delay property is compromised; the higher the filter order, the worse the group delay.

### Chebyshev type 1

Filters of this type are based on Chebyshev polynomials and have the fastest transition between the pass band and the stop band for a given ripple in the pass band and no ripples in the stop band.

## Chebyshev type 2

Filters of this type are defined by the start of the stop band and the stop band ripple. The filter has the fastest transition between the pass and stop bands given the stop band ripple and no ripple in the pass band.

## Notch

Notch filters are defined by a centre frequency and a  $q$  factor.  $q$  is the width of the stop band at the  $-3$  dB point divided by the centre frequency: the higher the  $q$ , the narrower the notch. Notch filters are sometimes used to remove mains hum, but if you do this you will likely need to set notches at the first few odd harmonics of the mains frequency. If you have a fairly constant coupling of mains noise into your signal, there are other ways to remove it that may cause much less signal degradation than notch filtering. There is a Hum Remove script on the CED web site that implements one method (select Spike2 scripts).

## Resonator

A resonator is the inverse of a notch. It is defined in terms of a centre frequency and a  $q$  factor.  $q$  is the width of the pass band at the  $-3$  dB point divided by the centre frequency: the higher the  $q$ , the narrower the resonance. Resonators are sometimes used as alternatives to a narrow bandpass filter.

## Non-filter bank commands

The `IIRBp()`, `IIRBs()`, `IIRLp()`, `IIRHp()`, `IIRReson()` and `IIRNotch()` commands are all independent and they each remember the last filter you created and the filter state after the last filtering operation. They have common operations to read back data and apply the filter to an array.

## Filter bank commands

The `IIRApply()`, `IIRComment$()`, `IIRCreate()`, `IIRInfo()` and `IIRName$()` commands use the IIR filter bank to generate and apply digital filters to channels of data.

### See also:

`FIRMake()`, `IIRApply()`, `IIRBp()`, `IIRBs()`, `IIRComment$()`, `IIRCreate()`, `IIRHp()`, `IIRInfo()`, `IIRLp()`, `IIRName$()`, `IIRNotch()`, `IIRReson()`

## Get IIR filter information

For the `IIRBp()`, `IIRBs()`, `IIRLp()`, `IIRHp()`, `IIRReson()` and `IIRNotch()` commands, you use a variant of the following form to return filter information:

```
Func IIRxxxx(get%{, arr[]{[]}});
```

`get%` The form of the command using this argument is used to read back information about the last filter created with this command. The argument can be:

- 1 `arr[]` is set to the impulse response of the filter. This is the response to an input of 1 followed by an infinite number of zeros assuming that all previous inputs were also zero. It is up to you to set a suitable length of the array. Each array data points corresponds with one sample interval. The return value is the magnitude of the largest value in `arr[]`. For example, if the impulse response ranged in values from  $-0.5$  to  $0.3$ ,  $0.5$  would be returned.
- 2 `arr[]` is set to the step response of the filter. The input to the filter is assumed to be an infinite number of zeros, followed by an infinite number of ones, and the response is taken from the moment where the input changes to a 1. It is up to you to set a suitable array length. The return value is the magnitude of the largest value in `arr[]`.
- 3 `arr[][]` is a matrix with  $r$  rows and 2 columns. The frequency response is returned as complex numbers in the columns; `arr[][0]` holds the real part and `arr[][1]` the imaginary part. The first row corresponds to a frequency of 0; the final row corresponds to a frequency of half the sampling rate. The frequencies are spaced as  $0.5/(r-1)$ . The more rows you set, the finer the frequency response. The return value is the maximum magnitude of the returned frequency response.
- 4 The same as 3 except that the results are returned as the amplitude response in column 0 and the phase response in column 1. If the real and imaginary parts of the response are  $r$  and  $i$ , `arr[][0]` holds  $\sqrt{r*r+i*i}$  and `arr[][1]` holds  $\text{atan}(i, r)$ . The return value is the

- maximum returned amplitude response.
- 5 Returns the number of filter coefficients ( $N+1$ ) to apply to the filter input values and fills in `arr[]` with these values. These correspond to the  $a_i$  in the filter expression. However, we return the values in reverse order as this makes them easier to use as a dot product with old values.  $N$  is the filter order for low pass and high pass, twice the order for band pass and band stop, and 2 for resonators and notch filters.
  - 6 Returns the number of filter coefficients ( $N+1$ ) to apply to the filter output values and fills in `arr[]` with these values. These correspond to the  $b_i$  in the filter expression. However, we return the values in reverse order to match the  $a_i$ . The final value is always  $-1.0$  (corresponding to  $b_n$ , which is not used when implementing the filter).  $N$  is the filter order for low pass and high pass, twice the order for band pass and band stop, and 2 for resonators and notch filters.
  - 7 Returns the filter gain  $G$  as defined in the filter expression.
  - 8 `arr[][]` is a matrix with  $N$  rows and 2 columns.  $N$  is the filter order for low pass and high pass, twice the order for band pass and band stop, and 2 for resonators and notch filters. The return value is the number of poles in the s-plane and the matrix is filled in with the poles as complex numbers with the real part in column 0 and the imaginary part in column 1; `arr[][0]` holds the real part and `arr[][1]` the imaginary part.
  - 9 The same as 8, but returning the s-plane zeros.
  - 10 The same as 8, but returning the z-plane poles.
  - 11 The same as 8, but returning the z-plane zeros.
  - 12 Returns a measure of filter stability, being the distance of the nearest pole to the unit circle. It is our experience that values greater than  $1e-12$  generate plausible filters. As the filter depends on a function of this distance, which is of the form (pole position-1), as the pole position approaches 1, the numerical accuracy of the result become significantly compromised (floating point numbers have around 15 significant digits of accuracy, all else being equal). You can improve the stability by reducing the order of the filter. Reporting stability was added at Spike2 version 7.09.

To read back information about a filter in the filter bank, use the `IIRInfo()` command.

#### See also:

`FIRMake()`, `IIRApply()`, `IIRBp()`, `IIRBs()`, `IIRHp()`, `IIRInfo()`, `IIRLp()`, `IIRNotch()`, `IIRReson()`

## Apply IIR filter to an array

This command applies the current filter set by one of the `IIRBp()`, `IIRBs()`, `IIRLp()`, `IIRHp()`, `IIRReson()` and `IIRNotch()` commands to an array of equally spaced data.

```
Func IIRBp(data[][, flags%[, save[]]);
Func IIRBs(data[][, flags%[, save[]]);
Func IIRLp(data[][, flags%[, save[]]);
Func IIRHp(data[][, flags%[, save[]]);
Func IIRReson(data[][, flags%[, save[]]);
Func IIRNotch(data[][, flags%[, save[]]);
```

`data` An array of data to filter.

`flags%` Optional, taken as 0 if omitted. It is the sum of:

- 1 To apply the filter backwards through the array. Applying a filter introduces a phase shift; running a filter forwards then backwards cancels the phase shift at the expense of a non-causal filter.
- 2 To treat `data[]` as a continuation of the last filtering operation (only do this if it really is a continuation, otherwise the results are nonsense). The filter state is saved separately for each filter type, but if you want to interleave use of the the same command between multiple data streams, you must use the `save[]` argument. If you apply the filter backwards, you must present the data blocks backwards.

`save[]` Optional. This is a real vector that preserves the state of the IIR filter for a data stream so that you can interleave continuous filtering with the same filter between multiple streams. The minimum size of the array depends on the filter type and order. It is  $2*\text{order}+2$  for `IIRLp()` and `IIRHp()`, 6 for `IIRReson()` and `IIRNotch()` and  $4*\text{order}+2$  for `IIRBp()` and `IIRBs()`.

If `save` is present and 2 is added to `flags%`, the filter state is loaded from it before filtering; it is always updated after filtering. You can simplify code by always having the continuous data flag set if you make sure that `save[]` is initialised to zeros before it is used for the first time.

IIR filters are recursive and causal, which means that the output of the filter at a particular time depends on all the data values up to that time. If you apply the filter as a one-off operation, this is not a problem. However, if you want to apply a filter in real time, or you want to filter a huge file and display your progress, you will need to apply the filter incrementally. To allow you to do this you can set a flag to say that the operation is a continuation of the previous operation. We save the filter "state" internally in each of the six IIRxxx() commands, so if you are working on a single data stream you can just set the continuous flag and the command will take care of this for you. However, if you want to apply the same type of filter to multiple data streams, you must supply storage space to save this internal state for each data stream. You cannot share the `save[]` vector between streams; either have different vector for each stream or use an array of two dimensions, one for the streams and one for saving the state.

```
var saveReson[6][2]; 'Space for two channels of Resonator state
...
Proc FilterTwoStreams(stream0[], stream1[], isFirst%);
if isFirst% then ArrConst(saveReson, 0) endif;
IIRReson(stream0[], 2, saveReson[][0]);
IIRReson(stream1[], 2, saveReson[][1]);
end;
```

#### See also:

FIRMake(), IIRApply(), IIRBp(), IIRBs(), IIRHp(), IIRLp(), IIRNotch(), IIRReson()

## IIRApply()

Applies a filter in the IIR filter bank to a waveform or RealWave channel in the current time view and places the result in a destination memory buffer or a disk-based channel. The output can be written as either a waveform or as a RealWave channel.

```
Func IIRApply(index%, dest%, srce%, sTime, eTime {,flags%});
```

`index%` Index of the filter in the filter bank to apply in the range -1 to 11

`dest%` The channel to hold the filtered waveform: either an unused disk channel, a memory channel with the same sampling frequency as `srce%` or 0 to create a compatible memory channel and place the filtered waveform in it. When a new channel is created, the channel settings are copied from the old channel.

`srce%` The source waveform or RealWave channel. There must be at least half the number of sampling coefficients worth of data points before `sTime` if the output is to start at `sTime`. Similarly, the channel must extend for the same number of data points beyond `eTime` if the output is to extend to `eTime`.

`sTime` Time to start the output of filtered data. There is no output for areas where there is no input data. If the filter has an even number of coefficients, the output is shifted by half a sample relative to the input.

`eTime` The end of the time range for filtered data.

`flags%` Optional, default value 0. The sum of the following flag values:

- 1 Optimises the destination channel scale and offset values to give the best possible representation of the output as 16-bit integers. For Waveform output channels, this doubles the processing time and existing data in the output channel that is not overwritten is also rescaled. If you do not rescale, the channel's scale and offset are unchanged. However, you run the risk of the output being clipped to the 16-bit range allowed for a waveform channel
- 2 Create a RealWave destination channel in place of a Waveform channel.

**Returns** The channel number that the output was written to or a negative error code. A negative error code is also returned if the user clicks Cancel from the progress bar that may appear during a long filtering operation or if `dest%` is a disk channel that is in use. Delete an existing channel with `ChanDelete(dest%)`.

#### See also:

The filter bank, ChanDelete(), IIRComment\$(), IIRCreate(), IIRBp(), IIRInfo(), IIRName\$()

## IIRBp()

This function creates and applies IIR (Infinite Impulse Response) band pass filters to arrays of data. You can run the filter forwards or backwards through the data.

```
Func IIRBp(data[]|0, lo, hi, order%, type%, ripple});
Func IIRBp(data[][, flags%[, save[]]);
Func IIRBp(get%[, arr[]{[]});
```

- data** An array of data to filter. If there are only 1 or 2 arguments, the last created band pass filter is used. Otherwise, the filter defined by the remaining arguments is used forwards. Replace `data` with 0 to create a filter without applying it.
- flags%** Optional, taken as 0 if omitted. Add 1 to run the filter backwards, 2 to treat `data[]` as a continuation of the last filtering operation.
- save[]** Optional. If present it must have a size of at least  $4 * \text{order}\% + 2$ . It is used to save the filter state to allow use with the continued data flag with multiple data streams.
- lo** The low corner frequency of the band stop filter as a fraction of the sample rate in the range 0.000001 to 0.499998. For Chebyshev type 2 filters, this is the point at which the attenuation reaches the `ripple` value, for all other filters this sets the -3 dB point.
- hi** The high corner frequency of the band pass filter as a fraction of the sampling rate in the range  $lo + 0.000001$  to 0.499999. For Chebyshev type 2 filters, this is the point at which the attenuation reaches the `ripple` value, for all other filters this sets the -3 dB point.
- order%** The order of the lowpass filter used as the basis of the design, in the range 1 to 10. The order of the filter implemented is  $\text{order}\% * 2$ . High orders ( $\text{order}\% > 7$ ) and narrow bands may cause inaccuracy in the filter. Narrow means that  $(hi - lo) / \sqrt{lo * hi}$  is less than 0.2, for example.
- type%** Set 0 for Butterworth, 1 for Bessel and 2 for Chebyshev type 1, 3 for Chebyshev type 2. The default value is 0 for a Butterworth filter.
- ripple** The desired pass band ripple in dB for Chebyshev type 1 filters (default 3 dB) or the desired minimum cut in the stop bands for Chebyshev type 2 filters (default 40 dB). The ripple value must be in the range 0.01 to 1000 dB.
- get%** The command variant with this argument returns information about the last filter you created with this command. See the discussion of IIR commands for details.
- arr** An optional vector or matrix used to return information about the last filter you created with this command. See the discussion of IIR commands for details.
- Returns** All forms of the command return negative numbers for errors. The forms that apply or create filters return 0 for success or 1 if a created filter has stability less than  $1e-12$ . The other command forms have their return values included in the description of the `get%` argument or return 0.

### See also:

More about IIR filters and commands, Get filter information, `FIRMake()`, `IIRBs()`, `IIRHp()`, `IIRLp()`, `IIRNotch()`, `IIRReson()`

## IIRBs()

This function creates and applies IIR (Infinite Impulse Response) band stop filters to arrays of data. You can run the filter forwards or backwards through the data.

```
Func IIRBs(data[]|0, lo, hi, order%, type%, ripple});
Func IIRBs(data[][, flags%[, save[]]);
Func IIRBs(get%[, arr[]{[]});
```

- data** An array of data to filter. If there are only 1 or 2 arguments, the last created band stop filter is used. Otherwise, the filter defined by the remaining arguments is applied forwards. Replace `data` with 0 to create a filter without applying it.
- flags%** Optional, taken as 0 if omitted. Add 1 to run the filter backwards, 2 to treat `data[]` as a continuation

of the last filtering operation.

- `save[ ]` Optional. If present it must have a size of at least  $4 * \text{order}\% + 2$ . It is used to save the filter state to allow use with the continued data flag with multiple data streams.
- `lo` The low corner frequency of the band pass filter as a fraction of the sample rate in the range 0.000001 to 0.499998. For Chebyshev filters, this is the point at which the attenuation reaches the `ripple` value, for other filters this sets the  $-3$  dB point.
- `hi` The high corner frequency of the band pass filter as a fraction of the sampling rate in the range  $lo + 0.000001$  to 0.499999. For Chebyshev type 2 filters, this is the point at which the attenuation reaches the `ripple` value, for all other filters this sets the  $-3$  dB point.
- `order%` The order of the lowpass filter used as the basis of the design, in the range 1 to 10. The order of the filter implemented is  $\text{order}\% * 2$ . High orders and narrow pass bands may lose numerical accuracy in the filter output.
- `type%` Set 0 for Butterworth, 1 for Bessel and 2 for Chebyshev type 1, 3 for Chebyshev type 2. The default value is 0 for a Butterworth filter.
- `ripple` The desired pass band ripple in dB for Chebyshev type 1 filters (default 3 dB) or the desired minimum cut in the stop band for Chebyshev type 2 filters (default 40 dB). The ripple value must be in the range 0.01 to 1000 dB.
- `get%` The command variant with this argument returns information about the last filter you created with this command. See the discussion of IIR commands for details.
- `arr` An optional vector or matrix used to return information about the last filter you created with this command. See the discussion of IIR commands for details.
- Returns All forms of the command return negative numbers for errors. The forms that apply or create filters return 0 for success or 1 if a created filter has stability less than  $1e-12$ . The other command forms have their return values included in the description of the `get%` argument or return 0.

**See also:**

More about IIR filters and commands, `Get filter information`, `FIRMake()`, `IIRBp()`, `IIRHp()`, `IIRLp()`, `IIRNotch()`, `IIRReson()`

## IIRComment\$()

This function gets and sets the comment associated with an IIR filter in the filter bank.

```
Func IIRComment$(index% {, new$});
```

`index%` Index of the filter in the filter bank to use in the range -1 to 11.

`new$` If present, sets the new comment.

Returns The previous comment for the filter at the index.

**See also:**

Filter banks, `IIRApply()`, `IIRInfo()`, `IIRName$()`

## IIRCreate()

This creates an IIR filter description and adds it to the filter bank.

```
Func IIRCreate(index%, type%, model%, order%, fr1{,fr2{,extra}});
```

`index%` Index of the filter in the filter bank in the range -1 to 11.

`type%` Sets the filter type as: 0=Low pass, 1=High pass, 2=Band pass, 3=Band stop.

`model%` Sets the filter model: 0=Butterworth, 1=Bessel, 2=Chebyshev type 1, 3=Chebyshev type 2, 4=Resonator.

`order%` Sets the filter order in the range 1-10. Resonators always set an order of 2.

- fr1** Sets the corner frequency for low pass, high pass filters, the centre frequency for resonators, and the low corner frequency for band pass and band stop filters.
- fr2** Sets the upper corner frequency for band pass/stop filters, otherwise ignored.
- extra** Sets the ripple for Chebyshev filters in the range 0.01 to 1000 and the Q factor for resonators in the range 1 to 10000.

Returns 0 if OK or a negative error code if the operation failed.

**See also:**

The filter bank, `IIRApply()`, `IIRComment$()`, `IIRCreate()`, `IIRName$()`

## IIRHp()

This function creates and applies IIR (Infinite Impulse Response) high pass filters to arrays of data. You can run the filter forwards or backwards through the data.

```
Func IIRHp(data[]|0, edge, order%{, type%{, ripple});
Func IIRHp(data[]{, flags%{, save[]});
Func IIRHp(get%{, arr[]{[]});
```

- data** An array of data to filter. If there are only 1 or 2 arguments, the last created high pass filter is used. Otherwise, the filter defined by the remaining arguments is applied forwards. Replace `data` with 0 to create a filter without applying it.
- flags%** Optional, taken as 0 if omitted. Add 1 to run the filter backwards, 2 to treat `data[]` as a continuation of the last filtering operation.
- save[]** Optional. If present it must have a size of at least  $2 * \text{order}\% + 2$ . It is used to save the filter state to allow use with the continued data flag with multiple data streams.
- edge** The corner frequency of the high pass filter as a fraction of the sample rate in the range 0.000001 to 0.499999. For Chebyshev filters, this is the point at which the attenuation reaches the `ripple` value, for other filters this sets the  $-3$  dB point.
- order%** The order of the filter in the range 1 to 10.
- type%** Set 0 for Butterworth, 1 for Bessel and 2 for Chebyshev type 1, 3 for Chebyshev type 2. The default value is 0 for a Butterworth filter.
- ripple** The desired pass band ripple in dB for Chebyshev type 1 filters (default 3 dB) or the desired minimum cut in the stop bands for Chebyshev type 2 filters (default 40 dB). The ripple value must be in the range 0.01 to 1000 dB.
- get%** The command variant with this argument returns information about the last filter you created with this command. See the discussion of IIR commands for details.
- arr** An optional vector or matrix used to return information about the last filter you created with this command. See the discussion of IIR commands for details.
- Returns All forms of the command return negative numbers for errors. The forms that apply or create filters return 0 for successor 1 if a created filter has stability less than  $1e-12$ . The other command forms have their return values included in the description of the `get%` argument or return 0.

**See also:**

More about IIR filters and commands, Get filter information, `FIRMake()`, `IIRBp()`, `IIRBs()`, `IIRLp()`, `IIRNotch()`, `IIRReson()`

## IIRInfo()

Retrieves information about an IIR filter in the bank.

```
Func IIRInfo(index%, &model%, &order%, &fr1{, &fr2{, &extra});
```

**index%** Index of the filter in the filter bank in the range -1 to 11.



- `model%` Returned as the filter model: 0=Butterworth, 1=Bessel, 2=Chebyshev type 1, 3=Chebyshev type 2, 4=Resonator.
- `order%` Returned as the filter order in the range 1-10. Resonators always return 2.
- `fr1` Returned as the corner frequency for low and high pass filters, as the low corner for band pass and band stop filters and as the centre frequency for resonators.
- `fr2` Returned as the upper corner frequency for band pass and band stop filters, otherwise set the same as `fr1`.
- `extra` Returned as the ripple for Chebyshev filters and as the Q factor for resonators.
- Returns The type of the filter as 0=Low pass, 1=High pass, 2=Band pass, 3=Band stop.

**See also:**

The filter bank, `IIRApply()`, `IIRComment$()`, `IIRCreate()`, `IIRName$()`

## IIRLp()

This function creates and applies IIR (Infinite Impulse Response) low pass filters to arrays of data. You can run the filter forwards or backwards through the data.

```
Func IIRLp(data[]|0, edge, order%, type%, ripple%});
Func IIRLp(data[][, flags%[, save[]]]);
Func IIRLp(get%[, arr[]{[]}]');
```

- `data` An array of data to filter. If there are only 1 or 2 arguments, the last created low pass filter is used. Otherwise, the filter defined by the remaining arguments is applied forwards. Replace `data` with 0 to create a filter without applying it.
- `flags%` Optional, taken as 0 if omitted. Add 1 to run the filter backwards, 2 to treat `data[]` as a continuation of the last filtering operation.
- `save[]` Optional. If present it must have a size of at least  $2 * \text{order}\% + 2$ . It is used to save the filter state to allow use with the continued data flag with multiple data streams.
- `edge` The corner frequency of the low pass filter as a fraction of the sample rate in the range 0.000001 to 0.499999. For Chebyshev filters, this is the point at which the attenuation reaches the `ripple` value, for other filters this sets the -3 dB point.
- `order%` The order of the filter in the range 1 to 10.
- `type%` Set 0 for Butterworth, 1 for Bessel and 2 for Chebyshev type 1, 3 for Chebyshev type 2. The default value is 0 for a Butterworth filter.
- `ripple` The desired pass band ripple in dB for Chebyshev type 1 filters (default 3 dB) or the desired minimum cut in the stop bands for Chebyshev type 2 filters (default 40 dB). The ripple value must be in the range 0.01 to 1000 dB.
- `get%` The command variant with this argument returns information about the last filter you created with this command. See the discussion of IIR commands for details.
- `arr` An optional vector or matrix used to return information about the last filter you created with this command. See the discussion of IIR commands for details.
- Returns All forms of the command return negative numbers for errors. The forms that apply or create filters return 0 for success or 1 if a created filter has stability less than  $1e-12$ . The other command forms have their return values included in the description of the `get%` argument or return 0.

**See also:**

More about IIR filters and commands, Get filter information, `FIRMake()`, `IIRBp()`, `IIRBs()`, `IIRHp()`, `IIRNotch()`, `IIRReson()`

## IIRName\$()

This function gets and/or sets the name of an IIR filter in the filter bank.

```
Func IIRName$(index% {, new$});
```

*index%* Index of the filter in the filter bank to use in the range -1 to 11.

*new\$* If present, sets the new name.

Returns The previous name of the filter at that index.

### See also:

Filter banks, IIRApply(), IIRComment\$(), IIRInfo()

## IIRNotch()

This function creates and applies IIR (Infinite Impulse Response) notch filters to arrays of data. You can run the filter forwards or backwards through the data. The gain of the notch filter is zero at the notch frequency and 1 at low and high frequencies.

```
Func IIRNotch(data[]|0, fr, q);
Func IIRNotch(data[]{, flags%{, save[]});
Func IIRNotch(get%{, arr[]{[]});
```

*data* An array of data to filter. If there are only 1 or 2 arguments, the last created notch filter is used. Otherwise, the filter defined by the remaining arguments is applied forwards. Replace *data* with 0 to create a filter without applying it.

*flags%* Optional, taken as 0 if omitted. Add 1 to run the filter backwards, 2 to treat *data[]* as a continuation of the last filtering operation.

*save[]* Optional. If present it must have a size of at least 6. It is used to save the filter state to allow use with the continued data flag with multiple data streams.

*fr* The frequency of the notch as a fraction of the sample rate in the range 0.000001 to 0.499999.

*q* The *q* factor for the notch in the range 1 to 10000; the higher the *q*, the narrower the notch. If *F10* and *Fhi* are the frequencies of the -3 dB points either side of the notch, *q* is  $fr / (Fhi - F10)$ . Try 100 as a starting point.

*get%* The command variant with this argument returns information about the last filter you created with this command. See the discussion of IIR commands for details.

*arr* An optional vector or matrix used to return information about the last filter you created with this command. See the discussion of IIR commands for details.

Returns All forms of the command return negative numbers for errors. The forms that apply or create filters return 0 for successor 1 if a created filter has stability less than 1e-12. The other command forms have their return values included in the description of the *get%* argument or return 0.

### See also:

More about IIR filters and commands, Get filter information, FIRMake(), IIRBp(), IIRBs(), IIRHp(), IIRLp(), IIRReson()

## IIRReson()

This function creates and applies IIR (Infinite Impulse Response) resonator filters to arrays of data. You can run the filter forwards or backwards through the data. The gain of the filter is 1 at the resonator frequency and zero at low and high frequencies.

```
Func IIRReson(data[]|0, fr, q);
Func IIRReson(data[]{, flags%{, save[]});
Func IIRReson(get%{, arr[]{[]});
```

- data** An array of data to filter. If there are only 1 or 2 arguments, the last created resonator filter is used. Otherwise, the filter defined by the remaining arguments is applied forwards. Replace `data` with 0 to create a filter without applying it.
- flags%** Optional, taken as 0 if omitted. Add 1 to run the filter backwards, 2 to treat `data[]` as a continuation of the last filtering operation.
- save[]** Optional. If present it must have a size of at least 6. It is used to save the filter state to allow use with the continued data flag with multiple data streams.
- fr** The centre frequency of the resonator as a fraction of the sample rate in the range 0.000001 to 0.499999.
- q** The *q* factor for the resonator in the range 1 to 10000; the higher the *q*, the narrower the resonance. If `Flo` and `Fhi` are the frequencies of the -3 dB points either side of the resonance, *q* is `fr/(Fhi-Flo)`. Try 100 as a starting point.
- get%** The command variant with this argument returns information about the last filter you created with this command. See the discussion of IIR commands for details.
- arr** An optional vector or matrix used to return information about the last filter you created with this command. See the discussion of IIR commands for details.
- Returns** All forms of the command return negative numbers for errors. The forms that apply or create filters return 0 for success or 1 if a created filter has stability less than 1e-12. The other command forms have their return values included in the description of the `get%` argument or return 0.

**See also:**

More about IIR filters and commands, Get filter information, `FIRMake()`, `IIRBp()`, `IIRBs()`, `IIRHp()`, `IIRLp()`, `IIRNotch()`

## Inkey()

This function is provided for compatibility with the MS-DOS version of Spike2. Do not use it in new scripts. It returns the ASCII code for the key pressed, or -1 if no key was pressed. In some cases Spike2 absorbs keystrokes, for example if you are sampling and the current window is the sampling time window all keystrokes are taken as markers.

**Func Inkey();**

**Returns** The key code or -1 if there is no pending key. The codes are:

|       |                     |        |                |
|-------|---------------------|--------|----------------|
| 1-31  | Ctrl+ABC...XYZ[\]^_ | 65-95  | ABC...XYZ[\]^_ |
| 32    | space               | 96     | `              |
| 33-47 | !"#\$%&'()*+,-./    | 97-126 | abc...xyz{ }~  |
| 48-64 | 0123456789:;<=>?@   | 127    | Rubout         |

**See also:**

`Interact()`, `KeyPress()`, `Toolbar()`, `ToolbarSet()`

## Input()

This function reads a number from the user. It opens a window with a message, and displays the initial value of a variable. You can limit the range of the result.

**Func Input(text\$, val{, low{, high{, pre%}}});**

- text\$** A string holding a prompt for the user. If the string includes a vertical bar, the text before the vertical bar is used as the window title.
- val** The initial value to be displayed for editing. If limits are given, and the initial value is outside the limits, it is set to the nearer limit.
- low** An optional low limit for the result. If `low >= high`, the limits are ignored.
- high** An optional high limit for the result.

**pre%** If present, this sets the number of significant figures to use to represent the number in the range 6 (the default) to 15.

**Returns** The value typed in. The function always returns a value. If an out-of-range value is entered, the function warns the user and a correct value must be given. When parsing the input, leading white space is ignored and the number interpretation stops at the first non-numeric character or the end of the string.

**See also:**

DlgReal(), DlgInteger(), DlgFont(), Input\$(), Inkey()

## Input\$()

This function reads user input into a string variable. It opens a window with a message, and displays a string. You can also limit the range of acceptable characters.

```
Func Input$(text$, edit${, maxSz${, legal$}});
```

**text\$** A string holding a prompt for the user. If the string includes a vertical bar, the text before the vertical bar is used as the window title.

**edit\$** The starting value for the text to edit.

**maxSz%** Optional, maximum size of the response string.

**legal\$** An string holding acceptable characters. **edit\$** is filtered before display. A hyphen indicates a range of characters. To include a hyphen in the list, place it first or last in the string. Upper and lower case characters are distinct. For upper and lower case characters and integer numbers use: "a-zA-Z0-9".

If this string is omitted, all printing characters are allowed, equivalent to " ~" (space to tilde). For simple use, the sequence of printing characters is:

```
space !"#%&'()*+,-./0123456789:;<=>?@
ABCDEFGHIJKLMNPOQRSTUVWXYZ[\]^_`
abcdefghijklmnopqrstuvwxyz{|}~
```

The order of extended or accented characters is system and country dependent.

**Returns** The result is the edited string. A blank string is a possible result.

**See also:**

DlgString(), DlgFont(), Input(), Inkey()

## InStr()

This function searches for a string within another string. This function is case sensitive.

```
Func InStr(text$, find${, index%});
```

**text\$** The string to be searched.

**find\$** The string to look for.

**index%** If present, the start character index for the search. The first character is index 1.

**Returns** The index of the first matched character, or 0 if the string is not found.

**See also:**

Chr\$(), DelStr\$(), LCase\$(), Left\$(), Len(), Mid\$(), Print\$(), ReadStr(), Right\$(), Str\$(), UCase\$(), Val()

## Interact()

This function provides a quick and easy way to interact with a user. It displays the interact toolbar at the top of the Spike2 window and pauses the script until a button or a key linked to a button is pressed. Cursors can always be dragged as we assume that they are one of the main ways of interacting with the data. You can limit the user actions when the bar is active.

```
Func Interact(msg$, allow% {,help {, lb1$ {,lb2$ {,lb3$...}}}) ;
```

**msg\$** A message to display in the message area of the toolbar. The message area competes with the button area. With many buttons, the text may not be visible.

**allow%** A number that specifies the actions that the user can and cannot take while interacting with Spike2. 0 allows the user to inspect data and position cursors in a single, unmovable window. The codes are shown in both decimal and hexadecimal format. The number is the sum of possible activity codes.

|      |        |                                     |
|------|--------|-------------------------------------|
| 1    | 0x0001 | Can change application              |
| 2    | 0x0002 | Can change the current window       |
| 4    | 0x0004 | Can move and resize windows         |
| 8    | 0x0008 | Can use File menu                   |
| 16   | 0x0010 | Can use Edit menu                   |
| 32   | 0x0020 | Can use View menu but not ReRun     |
| 64   | 0x0040 | Can use Analysis menu               |
| 128  | 0x0080 | Can use Cursor menu and add cursors |
| 256  | 0x0100 | Can use Window menu                 |
| 512  | 0x0200 | Can use Sample menu                 |
| 1024 | 0x0400 | No changes to y axis                |
| 2048 | 0x0800 | No changes to x axis                |
| 4096 | 0x1000 | No horizontal cursor channel change |

**help** This is either the number of a help item (CED internal use) or it is a help context string. This is used to set the help information that is presented when the user presses the F1 key. Set 0 to accept the default help. Set a string as displayed in the Help Index to select a help topic, for example "Cursors".

**lb1\$** These label strings create buttons, from right to left, in the tool bar. If no labels are given, one label is displayed with the text "OK". The maximum number of buttons is 17. Buttons can be linked to the keyboard using & and by adding a vertical bar followed by a key code to the end of the label. You can also set a tooltip. The format is "Label|code|tip". See the documentation for `label$` in the `ToolbarSet()` command for details.

**Returns** The number of the button that was pressed. Buttons are numbered in order, so `lb1$` is button 1, `lb2$` is button 2 and so on.

If a toolbar created by `Toolbar()` is present, it is hidden during the `Interact()` command and restored after `Interact()` returns.

### See also:

`Toolbar()`, `ToolbarSet()`

## K

### Keypress()

This function returns 1 if the `Inkey()` function would return a character, or 0 if it would not. This function and `Inkey()` are provided for compatibility with the MS-DOS version of Spike2 and are not recommended for new scripts.

**Func Keypress();**

Returns 1 if a key is ready to read, 0 if there is no key.

**See also:**

Inkey(), Interact(), Toolbar(), ToolbarSet()

**L****LastTime()**

This function finds the first item on a channel before a time. If a marker filter is applied to the channel, only data in the filter is visible. This function is for time views only.

**Func LastTime(chan%, time{,&val|code%[]{|data[]|data%[]|&data\$}});**

chan% The channel number in the view to use.

time The time to search before. Items at the time are ignored. To start a backward search that guarantees to iterate through all items, start at MaxTime(chan%)+1.

val Optional: for waveform channels it returns the waveform value. For event level channels, it is returned 0 if the transition is low to high, and 1 if the transition is high to low. If there is no event it returns the level at time; 0 for low, 1 for high.

code% This optional parameter is only used if the channel is a marker type (marker, RealMark, TextMark, WaveMark). This is an array with at least four elements that is filled in with the marker codes.

data Filled with data from RealMark and WaveMark channels. If there is insufficient data to fill it, unused entries are unchanged. An integer array can be used with WaveMark data to collect a copy of the 16-bit data that holds the waveform. If WaveMark data has multiple traces, use data[points%][traces%] to get real data and data%[points%][traces%] to get the integer data.

data\$ A string returned holding the text from a TextMark channel.

Returns The function returns either the time of the next item, or -1 if there are no more items to be found or a negative error code.

This function can be slow if run on a channel with a marker filter set and with the majority of events filtered out as Spike2 has to search for events that are in the filter.

**See also:**

ChanData(), MaxTime(), NextTime()

**LCase\$()**

This function converts a string into lower case.

**Func LCase\$(text\$);**

text\$ The string to convert.

Returns A lower cased version of the original string.

**See also:**

Asc(), Chr\$(), DelStr\$(), InStr(), Left\$(), Len(), Mid\$(), Print\$(), ReadStr(), Right\$(), Str\$(), UCase\$(), Val()

## Left\$()

This function returns the first *n* characters of a string.

```
Func Left$(text$, n);
```

*text\$* A string of text.

*n* The number of characters to extract.

Returns The first *n* characters, or all the string if it is less than *n* characters long.

### See also:

Asc(), Chr\$(), DelStr\$(), InStr(), LCase\$(), Len(), Mid\$(), Print\$(), ReadStr(), Right\$(), Str\$(), UCase\$(), Val()

## Len()

This function returns the length of a string or the size of a one dimensional array.

```
Func Len(text$);
```

```
Func Len(arr[]);
```

*text\$* The text string.

*arr[]* A one dimensional array. It is an error to pass in a two dimensional array.

Returns The length of the string or the array, as an integer.

You can find out the size of each dimension of a 2 dimensional array as follows:

```
proc something(arr[[[]]]) 'function passed a 2-d array
var n%; n% := Len(arr[[0]]); 'get size of first dimension
var m%; m% := Len(arr[0][[]]); 'get size of second dimension
```

The largest multi-dimensional array we support is 5-d, so here is the code for finding the sizes of all the dimensions of a 5-d array.

```
proc arrayMax(a[[[[[[]]]]])
var d1% := Len(a[[[0][0][0][0]]]);
var d2% := Len(a[0][[[0][0][0]]]);
var d3% := Len(a[0][0][[[0][0]]]);
var d4% := Len(a[0][0][0][[0]]);
var d5% := Len(a[0][0][0][0][[]]);
```

From version 7.11c you can pass an array with a 0 length, and the result is 0 rather than a fatal script error.

### See also:

Asc(), Chr\$(), DelStr\$(), InStr(), LCase\$(), Left\$(), Mid\$(), Print\$(), Right\$(), Str\$(), UCase\$(), Val()

## LinPred()

Linear prediction can be used to predict future (or past) data values based on a sequence of data values on the assumption that the data is statistically stationary. It can also be used to estimate power spectra using the Maximum Entropy or All Poles method. The command generates a set of coefficients that when applied to the previous *m* points, generate the next predicted point. Some of the explanation for this command relies on technical knowledge; see the references for more information. The command has the following variants:

```
Func LinPred(data[], mMax%{, limit{, out[]{, dir%{, stab%}}});
Func LinPred(0, out[]{, data[]}); 'Predict forward
Func LinPred(1, out[]{, data[]}); 'Predict backwards
Func LinPred(2, stab%); 'Check stability
Func LinPred(3, coef[]); 'Get the coefficients
Func LinPred(4, refl[]); 'Get reflection coefficients
Func LinPred(5, power[], frLo, frHi); 'Get estimated power spectrum
Func LinPred(6, poles[[[]]{, fr[]}); 'get poles and frequencies
```

- data** An array holding the data to be used to form the linear prediction coefficients or to be used to initialise a prediction based on existing coefficients.
- mMax%** The maximum order of the prediction, which is the number of previous points to use to predict each future point. The actual number may be less than this, depending on the value of the `limit` argument. You will generally want to use the smallest value of `m` that you can; values in the range 5-50 are common, but this does depend on the data. The value of `m` must be less than the number of data points and is generally much less. We have set an upper bound of 1024 on `mMax%` (this is higher than you are likely to need).
- limit** The algorithm to calculate the poles is an iterative procedure. The command holds an array of residual values that is initialised to the raw data. Each iteration subtracts data from the residual based on a normalised autocorrelation of the residual (in the range -1 to 1) at increasing lags, aiming to reduce the residual array to a list of zeros. We track a number that models the significance of the remaining data. This number is 1.0 at the start, and is multiplied by  $(1-ac^2)$  at each iteration, where `ac` is the autocorrelation, so the value decreases at each iteration unless `ac` is 0. If this value becomes less than `limit`, the iteration will stop. So set `limit` to 0 for no early stopping, setting a small value above  $1.0e-31$  may stop it early. The actual value of `m` used is returned by the first command variant.
- out** An array of output values predicted by the command. When predicting backwards, the data values are written into the `out` array so that the first data item in the array is the last predicted point (the oldest). When predicting forwards, the first item in the `out` array is the first predicted point.
- dir%** Optional argument to the setup command that sets the prediction direction. Set 0 or omit for a forward prediction, 1 for reverse.
- stab%** The linear prediction coefficients that are generated form a characteristic polynomial, and the roots of this polynomial are the positions of the poles in the z-plane. For the resulting filter to be stable (have an output that does not increase exponentially), the poles must lie inside (corresponding to decaying sinusoids) or on (corresponding to constant amplitude sinusoids) the unit circle. This algorithm should not produce poles outside the unit circle unless there are numerical accuracy problems (usually when `mMax%` is large). You can use this argument to check the stability of the result and to adjust the pole positions and recompute the coefficients by setting the values:
- 0 Do nothing to the poles.
  - 1 If a pole lies outside the unit circle, reflect it across the unit circle so that a growing sinusoid becomes a decaying sinusoid.
  - 2 If a pole lies outside the unit circle, move it onto the unit circle, corresponding to a constant sinusoid.
  - 3 Move all poles onto the unit circle. This produces an output that neither grows nor decays with time. However, this is very prone to numerical stability problems and is likely to be interesting only when `mMax%` is small.
- coef** An array to be filled in with coefficients. The array can be any size, but only the points corresponding with coefficients will be set.
- refl** An array to be filled with reflection coefficients. These values represent the proportion of the residual that was removed at each iteration. These are the values that are modified by the Levinson recursion to form the coefficients.
- power** An array to be filled in with estimated power spectrum components.
- frLo** A value in the range -0.5 to 0.5, being the fraction of the sampling frequency that the first bin of `power` will hold the estimated power spectrum for.
- frHi** A value in the range -0.5 to 0.5, being the fraction of the sampling frequency that the last bin of `power` will hold the estimated power spectrum for.
- poles** A matrix with the second dimension of size at least 2. `poles[n][0]` is returned holding the real component of the  $n^{\text{th}}$  pole, and `poles[n][1]` holds the imaginary component.
- fr** An array to be filled in with the frequencies that correspond to the pole positions, in the range -0.5 to 0.5 (fraction of the sampling rate).

## Setup

```
Func LinPred(data[], mMax%{, limit{, out[]{, dir%{, stab%}}});
```



This command must be used before any other as it calculates the initial set of coefficients. The command returns the number of coefficients that have been generated (this will be the same as `mMax%` if `limit` is 0). The command can be used to generate predicted data, or to set up the system for further `LinPred()` commands. The following example takes 1000 data points from near the start of a waveform channel, predicts the next 100 points and writes them to a memory channel.

```
const chan%=1; 'a waveform channel
var data[1000], out[100], sTime, bsz := BinSize(chan%);
var n% := ChanData(chan%, data, 100*bsz, MaxTime(), sTime);
var m% := LinPred(data, 20, 0, out, 0, 0); 'predict forwards
var mc% := MemChan(9, 0, bsz); 'new RealWave channel
MemSetItem(mc%, 0, sTime + n%*bsz, out[]); 'save the data
ChanShow(mc%); 'display it
```

The predicted data will not be the same as the actual data that follows the first 1000 points unless the first 1100 points are composed of the sum of constant amplitude sinusoids. The command forms a mathematical model of the data held in the `data[]` array based on the assumption that the spectral components are not changing with time (the signal is stationary) and that the data is modelled by a set of resonances. Unless you set `stab%` to 3, the result will usually decay with time.

### Predict forwards and backwards

```
Func LinPred(0, out[]{, data[]}); 'Predict forward
Func LinPred(1, out[]{, data[]}); 'Predict backwards
```

These two command versions fill the `out[]` array with data predicted by the coefficients established by the Setup version of the command. The setup command `dir%` argument will have set up the command to generate output that joins up with the original data array, or with any output if the setup command generated output. You can choose to continue generating output in the same direction, in which case you must NOT supply the `data[]` array, or you can supply a `data[]` array of at least the size of the number of coefficients to reset the prediction and you can then run forwards or backwards. By supplying a `data[]` array, you are not recalculating the coefficients, these remain unchanged; you are reloading the data points that are used with the coefficients to predict values. If you have `m` coefficients, when going forwards, the last `m` data points of `data[]` are used; when going backwards, the first `m` data points. For example, if we wanted to extend the previous example to predict the 100 data points that might have led up to the original 1000 points we could add the lines:

```
var back[100];
LinPred(1, back, data); ' predict back from start
MemSetItem(mc%, 0, sTime-100*bsz, back[]); ' save the data
```

If we had just used `LinPred(1, back)`; this would have caused an error as the previous use of the command was to go forwards. These command variants return 0.

### Check stability

```
Func LinPred(2, stab%); 'Check stability
```

This command variant is used to check that the poles of the characteristic polynomial lie within the unit circle. The command returns the distance of the pole furthest from the origin of the  $z$ -plane. This should be less than or equal to 1.0 for a stable set of coefficients. Stable, in this context, means that the predicted data does not grow exponentially. The algorithms we use should generate stable solutions, but poles can be generated outside the unit circle due to loss of numerical precision in the calculations. The `stab%` argument can be used to adjust the pole positions, as described above. If you are using this command to replace a short stretch of damaged data (like fixing a scratch in a record), you may want to predict both forwards and backwards across the damaged data, then mix the two predictions together. If the data used to generate the coefficients is not stationary, it will decay across the gap, in which case using `stab%` set to 3 will generate a result that maintains its amplitude, which may be what you require.

### Get coefficients

```
Func LinPred(3, coef[]); 'Get the coefficients
```

This function returns the number of coefficients and returns them in the `coef[]` array. Note that the first coefficient is the one that is multiplied with the most recent data point (when going forward), that is the coefficients run backwards compared to the data. Given an array `x[]` of data and coefficient `coef[]`, both of length `m%` points, the next forward and backward predictions are given by:

```
var i%, fwd:=0, rev:=0;
```

```
for i% := 0 to m%-1 do
 fwd += x[m%-i%-1] * coef[i%];
 rev += x[i%] * coef[i%];
next;
```

It is usually much simpler to use the forward and backward prediction versions of the command to do this. If you cannot do this, for example when you need to process several waveforms simultaneously, save the coefficients for each waveform (and reverse the order if predicting forwards), then you can use `ArrDot()` to predict each new point.

## Get reflection coefficients

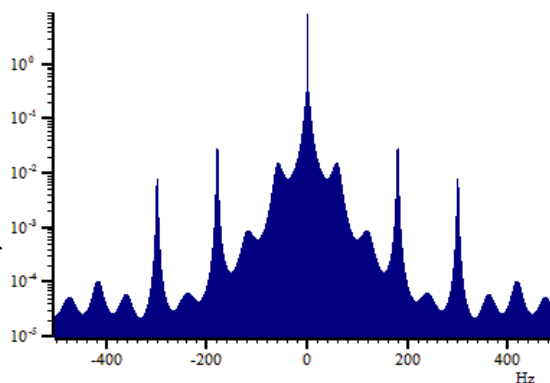
```
Func LinPred(4, refl[]); 'Get reflection coefficients
```

This function returns the number of reflection values that are available (the same value as the number of coefficients) and sets `refl` to the results of the auto-correlations done at each lag as the original data was analysed. The results are normalised such that the results lie in the range -1 to 1. The first value is the autocorrelation at a lag of 1 point, the next is for a lag of 2 points, and so on. The term reflection comes from the use of this technique in seismology.

## Power spectrum

```
Func LinPred(5, power[], frLo, frHi); 'Get estimated power spectrum
```

The variant generates an estimate of the power spectrum based on the representation of the original as the sum of a set of resonances. You should leave the `stab%` argument at 0 when using the command for this purpose. This method is particularly effective if you know that the data contains a small number of constant frequency sinusoids and can allow you to separate peaks that would merge into a single peak using the FFT. However, if your spectrum is not representable as the sum of a set of resonances, the result may be misleading unless `mMax%` is large enough so that the spectrum can be approximated. Unlike the FFT, where the resolution of the result depends on the number of points in the transform, here the resolution of a particular peak is determined by the number of bins and the frequency range that you set.



Typical Power spectrum

Make sure you set the bin width small enough so you do not miss a very narrow resonance. The frequencies are defined in terms of a fraction of the sampling rates from -0.5 to 0.5 (but note that -0.5 is the same frequency as 0.5); you will find that the power at frequency  $f$  is equal to the power at frequency  $-f$ .

The result is scaled so that the integral of the power from -0.5 up to (but not including) 0.5 is equal to the mean square of the values in the original `data[]` array used in the setup call. For example, we could extend the previous examples to display the power with:

```
const bins% := 10001; 'Bins in spectrum
var power[bins%]; 'Array to hold the power
LinPred(5, power, -0.5, 0.5); 'NB: -0.5 is same as 0.5
var rv% := SetResult(bins%, 1/(bsz*(bins%-1)), -0.5/bsz, "Power", "Hz");
ArrConst([], power);Optimise();WindowVisible(1); ' display the result
var MeanSq := ArrDot(data,data)/Len(data); 'Mean square of data
var MemSum := ArrSum(power[:bins%-1])/(bins%-1); 'Integral of power
Message("Mean sumSq = %g, SumPower = %g", MeanSq, MemSum);
```

The example image shows the power spectrum obtained by data with 60 Hz mains interference, leading to peaks at odd multiples of 60. If you try this with data constructed from pure sinusoids you will find that `MemSum < MeanSq`. The reason is that the purer the sinusoid, the higher and narrower the peak in the power spectrum; simple schemes for adding up equal width bins will not give an accurate result in this case. In the limiting case where the roots of the characteristic polynomial lie on the unit circle, the resonance has infinite amplitude, but zero width.

You might be puzzled about how this method can get much better frequency resolution than the FFT. The basic reason is that this method makes the assumption that the data continues in a *sensible way* outside the range of input points so as to preserve the auto-correlation between the data values. The FFT assumes that the data repeats exactly outside the initial range of data, leading to a limit on the frequency resolution.

## Get Poles and frequencies

```
Func LinPred(6, poles[][], fr[]); 'get poles and frequencies
```

In the  $z$ -plane, the coefficients are represented by a set of poles. You can use this variant to read back the positions of the poles as complex numbers. The poles are sorted in order by their real components, so the pole pairs for each imaginary root should be adjacent. You can also read back the frequencies (as a fraction of the sample rate in the range -0.5 to 0.5) at which the poles are located.

## Predicting data across a gap

If you are predicting data across a gap (a very common use of this function), you will find that your choice of `mMax%` is critical. If your data is periodic, for example a blood pressure signal and there are around  $p$  points in the period, then you will find that setting `mMax%` to  $p$  (the exact value is not critical) will usually work. If your data is the sum of a small number of sinusoids, you will find that a value of around twice the number of sinusoids may work. In other cases, if you are predicting across a gap of  $g$  points, then setting `mMax%` to  $g$  (again the exact value is not critical) may work.

It is a good idea to predict forwards and backwards across the gap, then merge the two predictions using some sort of weighting function that starts with all forwards prediction at the start and all backwards prediction at the end.

Remember that the mathematics makes the assumption that the data used for the prediction is stationary; that is that the statistical properties of the data (power spectra) remain constant with time. In many cases this will not be true; this leads to predicted data that decays away to the mean value.

The stability parameter should usually be 0. If you find that your predicted values are growing exponentially, then you can try setting the `stab%` argument to 1 or 2, which should stop this. However, the more normal problem is for the data to decay away. Setting `stab%` to 3 is a rather drastic act as it moves all the sinusoidal resonances that are used to model the power spectrum of the data onto the unit circle (converts them all into resonances with infinite  $Q$  and zero width). This may be useful if you have a small value of `mMax%` and you know that the data consists of constant sinusoids.

## References

Claerbout, Jon F. (1976). "Chapter 7 - Waveform Applications of Least-Squares." *Fundamentals of Geophysical Data Processing*. Palo Alto: Blackwell Scientific Publications. This has an explanation of the John P Burg algorithm that we implement.

Levinson recursion is a method for inverting a Toeplitz matrix in  $O(n^2)$  time, taking advantage of the symmetry of the matrix, and improving on the  $O(n^3)$  time of a general inversion ( $n$  is the order of the prediction). Symmetrical Toeplitz matrices come about as a natural consequence of the linear prediction equations. However, implementing the obvious equations often results in unsatisfactory solutions, and the Burg algorithm is a better approach that incorporates the iterative idea of the Levinson recursion, but calculates the autocorrelation in a different way that leads to stable solutions.

There is a good overview of linear prediction and the Maximum Entropy (All Poles) method of power spectrum estimation in *Numerical Recipes, The Art of Scientific Computing*, by Press, Flannery, Teukolsky and Vetterling.

## Ln()

This function calculates the natural logarithm (inverse of `Exp()`) of an expression, or replaces the elements of an array with their natural logarithms.

```
Func Ln(x|x[]{|[]...});
```

`x` A real number or a real array. Zero or negative numbers cause the script to halt with an error unless the argument is an array, when an error code is returned.

Returns When used with an array, it returns 0 if all was well, or a negative error code. When used with an expression, it returns the natural logarithm of the argument.

### See also:

`Abs()`, `ATan()`, `Cos()`, `Exp()`, `Frac()`, `Log()`, `Max()`, `Min()`, `Pow()`, `Rand()`, `Round()`, `Sin()`, `Sqrt()`, `Tan()`, `Trunc()`

## LnGamma()

This returns the natural logarithm of the gamma function  $\Gamma(x)$ , which exists for real values of  $x > 0.0$  and for negative non-integral values. For integral values of  $x$  less than or equal to 0,  $\Gamma(-n)$  is infinite.  $\Gamma(n+1)$  is the same as  $n!$  ( $n$  factorial) for integral values of  $n > 0$ . However, it increases very rapidly with  $x$ , reaching floating-point infinity when  $x$  is 172.62. To avoid this problem, the script returns the natural logarithm of the gamma function. The definition of the gamma function is:

$$\Gamma(a) = \int_0^{\infty} t^{a-1} e^{-t} dt$$

```
Func LnGamma(a);
```

a        A positive value. The script stops with a fatal error if this is negative.

Returns The natural logarithm of the Gamma function of a.

**See also:**

BinomialC(), GammaP()

## Log()

Takes the logarithm to the base 10 of the argument.

```
Func Log(x|x[]{|...});
```

x        A real number or a real array. Zero or negative numbers cause the script to halt with an error unless the argument is an array, when an error code is returned.

Returns With an array, this returns 0 if all was well or a negative error code. With an expression, this returns the logarithm of the number to the base 10.

**See also:**

Abs(), ATan(), Cos(), Exp(), Frac(), Ln(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan(), Trunc()

## LogHandle()

This returns the view handle of the log window (which always exists). You need this if you are to size the log window, or make it the current or front window, or to use the Edit menu commands to clear it. The log window is created by the application and is the destination for PrintLog(). The Log window is a simple text window. You can hide it by clicking in the go away box. You can clear the contents with the Edit menu Clear command.

```
Func LogHandle();
```

**See also:**

Print(), PrintLog(), View()

## M

### MarkEdit()

This changes the data stored in a marker at a particular time. You can get the data using LastTime() and NextTime().

```
Func MarkEdit(chan%, time, code%[]{|, data$|data[]|data%[]});
```

chan%    The marker, TextMark, WaveMark or RealMark channel to edit.

`time` The time of the marker (must match exactly).

`code%` Array of 4 marker codes (bottom 8 bits used) to replace markers in the channel.

`data` The data to replace the marker data. If you use integer data with a WaveMark, the bottom 16 bits of each integer replace the data. To update a WaveMark with multiple traces use a two-dimensional array, for example with 32 points and 4 traces use `var data[32][4];` to declare the array.

Returns 0 if a marker was edited, or -1 if no marker exists at `time`.

**See also:**

`LastTime()`, `MarkInfo()`, `MarkMask()`, `MarkSet()`, `MarkShow()`, `NextTime()`

## MarkInfo()

This function is used to get information on the extended marker types (TextMark, WaveMark and RealMark).

```
Func MarkInfo(chan% {,&pre% {,trace%}});
```

`chan%` The channel to get the information from.

`pre%` If present, returned as the number of pre-peak points for WaveMark data.

`trace%` If present, returned as the number of traces (electrodes) in the WaveMark data.

Returns For WaveMark data it returns the number of waveform points, for TextMark data it returns the maximum string length and for RealMark data it returns the number of reals attached to each marker. For all other channel types it returns 0.

**See also:**

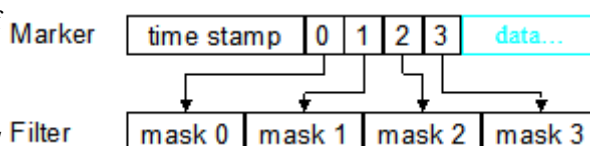
`LastTime()`, `MarkEdit()`, `MarkMask()`, `MarkSet()`, `MarkShow()`, `NextTime()`

## MarkMask()

This function sets the mask for a marker, WaveMark, TextMark or RealMark channel. Each data item in one of these channels has four marker codes. Each code has a value from 0 to 255. Each data channel (and duplicated channel) has its own marker filter that determines the visible data items. A marker filter has four masks, one for each of the four marker codes. For each mask, you can specify which codes are wanted. There are two marker filter modes. In the diagrams, a marker data item is represented as a time stamp, four marker codes and data values that depend on the marker type.

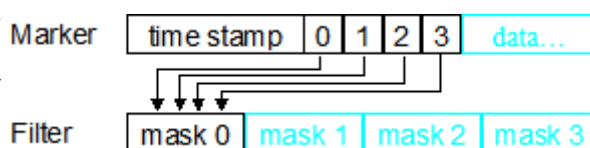
### Mode 0 (AND)

A marker data item is allowed through the mask if each of the four codes in the data item is present in the corresponding mask. We think of this as *and* mode because for the filter to pass the marker, marker code 0 must be in mask 0 *and* marker code 1 in mask 1 *and* marker code 2 in mask 2 *and* marker code 3 in mask 3. In this mode, masks 1, 2 and 3 are usually set to accept all codes and masking is used for layer 0.



### Mode 1 (OR)

A marker data item is allowed through the mask if any of the four marker codes is present in mask 0. A code 00 is only accepted for the first of the four marker codes. Masks 1 to 3 are ignored. We think of this as *or* mode because marker code 0 *or* 1 *or* 2 *or* 3 must be present in mask 0 for the filter to pass the marker for display or analysis. This mode can be used with spike shape data (WaveMark) where two spikes have collided and the marker represents a spike of two different templates.



There are four command variants, the first reports if the mask is active, the second sets the filter codes, the third sets the filter mode and the fourth allows you to read and set the entire mask:

```
Func MarkMask(chan%);
Func MarkMask(chan%, layer%, set%, code%|code$ {,code%|code$...});
Func MarkMask(chan%, mode%);
Func MarkMask(chan%, mask%[]{}{, write%});
```

chan% The channel number to work on. If unsuitable, MarkMask() returns a negative error code.

layer% The layer of the mask in the range 0 to 3 or -1 for all layers.

set% 1 to include codes in the mask, 0 to exclude codes or -1 to invert the mask. Inverting a mask changes all included codes to excluded and vice versa.

code% A number in the range 0 to 255 setting a code to include or exclude. You can specify more than one code at a time. -1 is also allowed, meaning all codes.

code\$ Each character in the string is converted to its ASCII value, and used as a code.

mode% The variant with two arguments returns the current mode of the marker filter and optionally sets the mode of matching as 0 or 1 or -1 for no change.

mask% This is an array that is usually declared as mask%[256][4] or mask%[256] that maps onto the mask.

write% If this optional argument is omitted or zero, the mask% array is filled with data. If it is 1, the mask data is copied to the mask.

Returns A negative error code or 0 except the first variant which returns 1 if the current mask is active, 0 if not active.

### Report active state

```
Func MarkMask(chan%);
```

The first variant was added at Spike2 version 8.01 and reports if the current mask is active. An active mask is one that could prevent some items being displayed. However, being active does NOT mean that items are filtered out as this depends on the data in the channel.

### Set filter codes

```
Func MarkMask(chan%, layer%, set%, code%|code$ {,code%|code$...});
```

The second variant allows you to set codes, either individually or by layer. A common requirement is to allow all markers to be used. This is achieved by:

```
MarkMask(chan%, -1, 1, -1); 'Set all layers to 1 for all codes
```

To fill or empty or invert a complete layer use:

```
MarkMask(chan%, layer%, set%, -1); 'Apply set% to entire layer
```

This example sets the keyboard marker channel mask (channel 31) to show only markers 0 and 1 (start and stop recording markers) and key presses for A, B, C and D:

```
MarkMask(31, 0); 'set mode 0
MarkMask(31, -1, 1, -1); 'include everything (reset)
MarkMask(31, 0, 0, -1); 'exclude everything in layer 0
MarkMask(31, 0, 1, 0, 1, "ABCD"); 'include the codes we want
```

You can use this command together with ChanDuplicate() to split a marker channel into several channels based on marker codes.

### Set or get the filter mode

```
Func MarkMask(chan%, mode%);
```

This sets the filter mode to AND (mode%=0) or OR (mode%=1) and returns the original mode. To get the mode without a change use MarkMask(chan%, -1).

### Set or get the mask

```
Func MarkMask(chan%, mask%[]{}{, write%});
```

This command copies an array of mask values to the filter or reads the filter into an array. mask% is array that is usually declared as mask%[256][4] or mask%[256] that maps onto the mask. The second dimension sets

the layer. A one-dimensional array maps onto layer 0. When reading, elements are set to 0 or 1. When writing, zero element clear and non-zero elements set the corresponding item in the mask. If the array size does not match 256 by 4, data is transferred between array items that map onto the mask.

**See also:**

`ChanDuplicate()`, `LastTime()`, `MarkEdit()`, `MarkInfo()`, `MarkSet()`, `MarkShow()`, `NextTime()`

## MarkSet()

This sets the marker codes of a marker, WaveMark, TextMark or RealMark channel in a time range. If a marker filter mask is set, only data that is passed by the mask is changed.

```
Func MarkSet(chan%, sT, eT, code%[]);
Func MarkSet(chan%, sT, eT, c0%{, c1%{, c2%{, c3%}}});
```

`chan%` The channel to process.

`sT, eT` The time range to process.

`code%` An array of 4 integers holding the new marker codes in the range 0 to 255 or -1 for a code that is unchanged.

`c0-c3%` One to four marker codes as an alternative to `code%[]`. Use values 0 to 255 to change a code and -1 (or omit the value) to leave a code unchanged.

Returns The number of markers that were changed.

**See also:**

`LastTime()`, `MarkEdit()`, `MarkMask()`, `MarkShow()`, `NextTime()`

## MarkShow()

This sets which of the four marker codes to display for one or more marker channels and the format used when displaying the marker code in a time view. Setting the default draw mode with `DrawMode(chan%, 0)` resets the displayed code and drawing mode to 0. This is equivalent to the Marker code and Hex only fields in the channel Draw Mode dialog.

```
Func MarkShow(cSpc{, code%{, mode%}});
```

`cSpc` A channel specifier or -1 for all, -2 for visible or -3 for selected channels.

`code%` This optional argument sets which of the 4 marker codes to display in the range 0 to 3, or -1 or -2 for no change (-2 returns the current drawing mode).

`mode%` Omit or set to -1 for no change. Otherwise set 1 to display all codes as two hexadecimal digits, or 0 to show codes 0x20 to 0x7e as the character equivalent.

Returns The original marker code to display (0-3) or, if `code%` is -2, the drawing mode.

**See also:**

`DrawMode()`, `LastTime()`, `MarkEdit()`, `MarkMask()`, `NextTime()`

## MATDet()

This calculates the determinant of a matrix (a two dimensional array).

```
Func MATDet(mat[][]);
```

`mat` A two dimensional array with the same number of rows and columns.

Returns The determinant of `mat` or 0.0 if the matrix is singular.

**See also:**

ArrAdd(), MATInv(), MATMul(), MATTrans()

## MATInv()

This inverts a matrix (a two dimensional array) and optionally returns the determinant.

```
Func MATInv(inv[][] {,src[][]{, &det}});
```

**inv** A two dimensional array to hold the result. If **src** is omitted, **inv** is replaced by its own inverse. The number of rows and columns of **inv** must be the same.

**src** If present, the matrix to invert. The numbers of rows and columns of this two dimensional array must be at least as large as **inv**.

**det** If present, returned holding the determinant of the inverted matrix.

Returns 0 if all was OK, -1 if the matrix was singular or very close to singular.

**See also:**

ArrAdd(), MATMul()

## MatLab Script Support

If you have MATLAB™ installed on your computer and selected the MATLAB script option when installing Spike2, the `MatLabXxx()` script commands let you start a MATLAB process to use as a computational engine. This process can have a visible window allowing some user interaction; it is not a full MATLAB workspace and is separate from any normally opened MATLAB workspaces that you use. You can transfer script variables and arrays (but not arrays of strings) to it, command it to process your data, then move results back into the script language.

### Example

This example can be used to check that the script support is working. It illustrates use of all the functions and most of the data transfer options.

```
var f%, a, b, c, m%, n%;

f% := MatLabOpen(); ' Open MATLAB
if (f% < 0) then Message("Open failed %d", f%); halt endif;

f% := MatLabShow(); ' Use MatLabShow
if (f% <> 0) then Message("Show initial state %d", f%) endif;
f% := MatLabShow(1);
if (f% <> 0) then Message("Show 1: state %d", f%) endif;
f% := MatLabShow(0);
if (f% <> 1) then Message("Show 0: state %d", f%) endif;
f% := MatLabShow();
if (f% <> 0) then Message("Show final state %d", f%) endif;
MatLabShow(1);

' Test put and get of a simple variable
a := 2.5;
f% := MatLabPut("var_a", a);
if (f% <> 0) then Message("MatLabPut var_a result %d", f%) endif;
f% := MatLabGet("var_a", c);
if (f% <> 0) then Message("MatLabGet var_a result %d", f%) endif;
if (c <> 2.5) then Message("get value after put 2.5 is %g", c) endif;

' Test put of variable, eval then get the result
b := 1.5;
f% := MatLabPut("var_b", b);
if (f% <> 0) then Message("MatLabPut var_b result %d", f%) endif;
f% := MatLabEval("var_b = var_b + var_a");
if (f% <> 0) then Message("MatLabEval result %d", f%) endif;
f% := MatLabGet("var_b", c);
if (f% <> 0) then Message("MatLabGet result %d", f%) endif;
```



```

if (c <> 4) then Message("MatLabGet of eval(1.5 + 2.5): %g", c) endif;

' Test put and get of 1D arrays.
var ar1%[4],ar2d%[4][1],ar2dt%[1][4],ar2%[4],ar3%[6]; ' Test with 1D arrays
ar1%[0] := 12; ar1%[1] := 11; ar1%[2] := 120; ar1%[3] := 1200;
f% := MatLabPut("tar",ar1%[]); ' Trivial put & get
if (f% <> 0) then Message("Put ar1 result %d", f%) endif;
f% := MatLabGet("tar",ar2%[]);
if (f% <> 4) then Message("Get ar2 result %d", f%) endif;
Compare1D("transfer 1D", ar1%, ar2%);

f% := MatLabGet("tar", ar2d%); ' Now get into nxl array
if (f% <> 1) then Message("Get ar2d result %d, should be 1", f%) endif;
Compare1D("Transfer to nxl", ar1%, ar2d%[0]);

var r$;
MatLabEval("tart = tar'", r$); ' Next get the transposed data
f% := MatLabGet("tart",ar2%[]);
if (f% <> 4) then Message("Get tart to ar2 result %d", f%) endif;
Compare1D("1D transposed", ar1%, ar2%);

f% := MatLabGet("tart",ar2dt%[][]); ' Now get into lxn array
if (f% <> 4) then Message("Get ar2dt result %d", f%) endif;
Compare1D("Transfer to lxn", ar1%, ar2dt%[0][1]);
f% := MatLabGet("tart",ar3%[]); ' get into larger 1D array
if (f% <> 4) then Message("Get tart to ar3 result %d", f%) endif;
Compare1D("transfer 1D to larger", ar3%[:4], ar2%);

var stri$; ' Test with a string variable
f% := MatLabPut("string","abc");
if (f% <> 0) then Message("Put string result %d", f%) endif;
f% := MatLabGet("string",stri$);
if (f% <> 0) then Message("Get string result %d", f%) endif;
if (stri$ <> "abc") then Message("sent abc, got %s", stri$) endif;

' Now tests on nxn arrays. We check put, simple get, get into
' array with last dimension bigger and get after transpose
var arr[3][3];
arr[0][0] := 1; arr[0][1] := 2; arr[0][2] := 3;
arr[1][0] := 4; arr[1][1] := 5; arr[1][2] := 6;
arr[2][0] := 7; arr[2][1] := 8; arr[2][2] := 9;
f% := MatLabPut("var_arr", arr); ' Copyto MATLAB
if (f% <> 0) then Message("MatLabPut arr result %d", f%) endif;
var arrg[3][3]; ' Simple get test
f% := MatLabGet("var_arr", arrg);
if (f% <> 3) then Message("Get arrg result %d", f%) endif;
Compare2D("Get 3x3 to 3x3", arr, arrg);

var arrb[3][6]; ' Get to larger array test
f% := MatLabGet("var_arr", arrb);
if (f% <> 3) then Message("Get arrb result %d", f%) endif;
Compare2D("Get 3x3 to 3x6", arr, arrb);

MatLabEval("var_arrt = var_arr'", r$); ' Now get after transpose
f% := MatLabGet("var_arrt", arrg);
if (f% <> 3) then Message("Get arrg result %d", f%) endif;
MatTrans(arr); ' Transpose data to match
Compare2D("3x3 transpose", arr, arrg); ' Check

MatLabClose();
halt;

Proc Compare1D(test$, a%[], b%[])
var bad% := 0, m%, n1%;
n1% := Len(a%);
for m% := 0 to n1%-1 do
 if (a%[m%] <> b%[m%]) then
 PrintLog("%s: 1D mismatch at %d, got %g, expected %d\n",
 test$, m%, b%[m%], a%[m%]);
 bad% += 1;
 endif;
next;
if (bad% > 0) then Message("%s: %d bad values", test$, bad%) endif;
end;

```

```

Proc Compare2D(test$, a[[]], b[[]])
var m%, n%, n1%, n2%, bad% := 0;
n1% := Len(a[0]); n2% := Len(a[0][0]);
for m% := 0 to n2%-1 do
 for n% := 0 to n1%-1 do
 if (a[n%][m%] <> b[n%][m%]) then
 PrintLog("%s Mismatch at %d %d, got %g, expected %g\n",
 test$, n%, m%, b[n%][m%], a[n%][m%]);
 bad% += 1;
 endif;
 next;
next;
if (bad% > 0) then Message("%s: %d bad values", test$, bad%) endif;
end;

```

**See also:**

MatLabOpen(), MatLabClose(), MatLabPut(), MatLabGet(), MatLabEval(), MatLabShow()

## MatLabOpen()

This opens a connection to an invisible MATLAB command window. Other MatLabXxx() commands fail if there is no connection. Making a connection takes a detectable time; it is inadvisable to open and close the connection many times.

```
Func MatLabOpen({mode%});
```

mode% If mode% is absent or zero, the connection is shared, allowing other applications to use the same command window. Set mode% to 1 for unshared, for exclusive use by the script.

Returns 0 if the connection was opened successfully, -1 if the function failed.

**See also:**

MatLabClose(), MatLabPut(), MatLabGet(), MatLabEval(), MatLabShow(), script example

## MatLabClose()

This function closes a connection that was created using MatLabOpen().

```
Proc MatLabClose();
```

**See also:**

MatLabOpen(), MatLabPut(), MatLabGet(), MatLabEval(), MatLabShow(), script example

## MatLabPut()

This function takes a script variable and puts it into the connected MATLAB command window.

```
Func MatLabPut(name$, v|v$|v%[1][1...]|v[1][1...]{, as%});
```

name\$ the name of a MATLAB variable to create or overwrite. Names may use only alphanumeric characters plus underscore '\_' and must start with an alphabetic character.

v An expression or variable holding data to move to MATLAB. The created variable type is set by the type of v and the as% argument. All types except string arrays are supported.

as% Optional. Sets the type of the MATLAB variable. If v is a real value or array, as% can be 4 or 8 to specify single- or double-precision real data. If v is an integer value or array, as% can be 1, 2, 4 or 8 for 1, 2, 4 and 8-byte signed integer data and -1, -2, -4 and -8 for the corresponding unsigned integer data. For 1, 2 and 4-byte integer data only the lower 1, 2 or 4 bytes are used. If as% is omitted, it is taken as 8 for a real value and 4 for an integer value.

For string data, a string variable is created in the workspace and as% is ignored.

For integer and real data, either a single variable or a vector is created in the MATLAB workspace. If

an array is used then the dimensions of the variable created in the workspace match the script array: a 1-dimensional script array length  $n$  creates an  $n \times 1$  vector, a 2 dimensional script array `arr[m][n]` creates an  $m \times n$  matrix and so forth.

Returns 0 if the data was successfully transferred and -1 if it was not.

#### See also:

`MatLabOpen()`, `MatLabClose()`, `MatLabGet()`, `MatLabEval()`, `MatLabShow()`, [script example](#)

## MatLabGet()

This function copies a variable from the MATLAB command window into a script variable.

```
Func MatLabGet(name$, &v%|&v|&v$|v%[1]{[1...]|v[1]{[1...]});
```

`name$` the name of the MATLAB variable to copy. If the variable does not exist the function returns -1 and script execution continues. MATLAB variables that are structures, cell arrays, functions or logical (BOOLEAN) data are not supported. Of the remaining data types (character and numeric values), the allowed types vary with the type of script language variable supplied.

`v$` For a string variable, the MATLAB variable must be a simple string (a 1-dimensional array of characters). String arrays are not supported.

`v% v` For non-array integer and real variables, the MATLAB variable must be a numeric value. Integer and floating-point data are automatically converted, floating point values put into an integer variable are truncated.

`v[ ]` For real and integer script arrays, the MATLAB variable must match the number of dimensions and type (integer or real). All dimension sizes except that last must also match. The last dimension size must be less than or equal to the size of the last script dimension. We make an exception for a script vector which matches both  $n \times 1$  and  $1 \times n$  MATLAB variables. MATLAB floating-point types are converted to script reals and MATLAB integer types are converted to script integers. The return value is the number of items in the last dimension of the script array that was filled.

Returns -1 if the variable was not found in MATLAB or a type match error, otherwise it returns 0 for a simple variable or the size of the last dimension of the script array that was filled.

### Command changes

At Spike2 revision 7.09, the command was changed to be more tolerant when returning arrays. The table shows which Spike2 variables are compatible with which MATLAB variables as long as  $n \leq m$ .

| Spike2                         | MATLAB                                                           |
|--------------------------------|------------------------------------------------------------------|
| <code>s1[m]</code>             | <code>m1(n)</code> , <code>m1(n,1)</code> , <code>m1(1,n)</code> |
| <code>s2[a][m]</code>          | <code>m2(a,n)</code>                                             |
| <code>s3[a][b][m]</code>       | <code>m3(a,b,n)</code>                                           |
| <code>s4[a][b][c][m]</code>    | <code>m4(a,b,c,n)</code>                                         |
| <code>s5[a][b][c][d][m]</code> | <code>m5(a,b,c,d,n)</code>                                       |

The return value was also changed. An error is still flagged by a -1 return, but success is no longer indicated by a return value of 0. The return value is now 0 for simple variables and  $n$  (as in the table) for vectors and multi-dimensional arrays. If you have a script that used to work before version 7.09, look for code like:

```
if (MatLabGet(...) = 0 then
```

and change `= 0` to `>= 0` (so it works with old and new versions of the command).

#### See also:

`MatLabOpen()`, `MatLabClose()`, `MatLabPut()`, `MatLabEval()`, `MatLabShow()`, [script example](#)

## MatLabEval()

Requests the MATLAB command window to execute an arbitrary command and gets the result as a string. Typical use: set data with `MatLabPut()`, process with `MatLabEval()`, get the result with `MatLabGet()`.

```
Func MatLabEval(cmd$ {,&resp$});
```

`cmd$` This is the command string that is sent to the command window, for example "x = a + b".

`resp$` Optional. If present, it is set to the command window response. For example, it is set to an error message if the command fails. We limit the response length to 511 characters.

Returns 0 if `cmd$` was successfully passed to the command window and -1 if it was not.

### See also:

`MatLabOpen()`, `MatLabClose()`, `MatLabGet()`, `MatLabPut()`, `MatLabShow()`, [script example](#)

## MatLabShow()

This command retrieves the visible or hidden status of the MATLAB command window opened by `MatLabOpen()` and optionally changes it.

```
Func MatLabShow({show%});
```

`show%` Optional. If provided, sets the new visible state. Set 1 to show the window and 0 to hide it.

Returns the command window visibility at the time the call was made.

### See also:

`MatLabOpen()`, `MatLabClose()`, `MatLabGet()`, `MatLabPut()`, `MatLabEval()`, [script example](#)

## MATMul()

This function multiplies matrices and/or vectors. In matrix terms, this evaluates  $A = BC$  where  $A$  is an  $m$  rows by  $n$  columns matrix,  $B$  is an  $m$  by  $p$  matrix and  $C$  is a  $p$  by  $n$  matrix. Vectors of length  $v$  are treated as a  $v$  by 1 matrix.

```
Proc MATMul(a[][], b[][], c[]{});
```

`a` An  $m$  by  $n$  matrix of reals or a vector of length  $m$  ( $n$  is 1) to hold the result.

`b` An  $m$  by  $p$  matrix or a vector of length  $m$  ( $p$  is 1).

`c` A  $p$  by  $n$  matrix or a vector of length  $p$  ( $n$  must be 1).

If you pass any of `a`, `b` or `c` as a vector, they are treated as a  $n$  by 1 matrix, where  $n$  is the length of the vector. Use the `trans()` operator to convert a vector to a 1 by  $n$  matrix.

### See also:

`trans()` operator, `ArrMul()`, `MATInv()`

## MATSolve()

This function solves the matrix equation  $Ax = y$  for  $x$ , given  $A$  and  $y$ . Both  $x$  and  $y$  are vectors of length  $n$  and  $A$  is an  $n$  by  $n$  matrix.

```
Func MATSolve(x[], a[][], y[]);
```

`x` A one dimensional real array of length  $n$  to hold the result.

`a` A two dimensional ( $n$  by  $n$ ) array of reals holding the matrix.

`y` A one dimensional real array of length  $n$ .

Returns The functions returns 0 if all is OK or -1 if a is a singular matrix.

**See also:**

`ArrMul()`, `MATInv()`

## MATTrace()

This evaluates the trace of a matrix, that is the sum of the diagonal of the matrix. If the matrix is not square, for example `arr[i][j]` and the smaller dimension is of size `n`, the trace is of the `arr[:n][:n]`.

**Func** `MATTrace(arr[][])`

`arr` A matrix (two-dimensional array) that is either square, or that is treated as square using the smaller dimension. The array can be either real or integer. The result is always a real value.

Returns The trace of the matrix, that is the sum of the diagonal elements.

This is equivalent to `ArrSum(diag(arr))`;

## MATTrans()

This transposes a matrix (a two dimensional array), swapping the rows and columns. This procedure physically moves the data, unlike the `trans()` or ``` operator, which remaps the matrix without moving any data. It is usually much more efficient to use `trans()`.

**Proc** `MATTrans(mat[][]{, src[][]});`

`mat` A `m` by `n` matrix returned holding the transpose of `src`. If `src` is omitted, `m` must be equal to `n` and the rows and columns of `mat` are swapped.

`src` Optional, a `n` by `m` matrix to transpose.

**See also:**

`trans()` operator, `ArrAdd()`, `MATMul()`

## Max()

This function returns the index of the maximum value in an array, or the maximum of several real and/or integer variables.

**Func** `Max(arr[]|arr%[]|val1 {,val2 {,val3...}});`

`arr` A real or integer array.

`valn` A list of real and/or integer values to scan for a maximum.

Returns The maximum value or array index of the maximum.

**See also:**

`Abs()`, `ATan()`, `Cos()`, `Exp()`, `Frac()`, `Ln()`, `Log()`, `Min()`, `MinMax()`, `Pow()`, `Rand()`, `Round()`, `Sin()`, `Sqrt()`, `Tan()`, `Trunc()`, `XYRange()`

## Maxtime()

In a time view, this returns the maximum time in seconds in the file or in a channel, or the current sample time during sampling. In a result view, it returns the number of bins.

**Func** `MaxTime({chan%});`

`chan%` Optional channel number for time views, ignored in result views. If present, the function gets the maximum time in the channel ignoring any marker filter. If there is no data in the channel, the return

value is -1.

Returns The value returned is negative if the channel does not exist. If the current view is of the wrong type the script stops with an error. If there is no data in the channel or file, the return value is -1.

To find the time of the last item in the marker filter on a channel with a marker filter set:

```
time := LastTime(chan%, MaxTime()+BinSize());
```

With a marker filter set, this will search the data backwards from the end to find a marker in the filter. `MaxTime()` returns the last valid time in the file on all channels. The `+BinSize()` is because `LastTime()` finds data before the search time.

**See also:**

`Len()`, `LastTime()`, `NextTime()`, `Seconds()`

## MeasureChan()

This function adds or changes a measurement channel in an XY view created with `MeasureToXY()` using the settings previously defined by using `MeasureX()` and `MeasureY()`. The XY view must be the current view. This command implements some of the functionality of the XY plot setting dialog.

```
Func MeasureChan(chan%, name${, pts%});
```

`chan%` This is 0 to add a new channel or the number of an existing channel to change settings. `MeasureToXY()` creates an XY view with one channel, so you will usually call this function with `chan%` set to 1. You can have up to 32 measurement channels in the XY view.

`name$` This sets the name of the channel and can be up to 9 characters long.

`pts%` Sets the maximum number of points for this channel, if omitted or set to zero then all points are used. When a points limit is set and more points are added, the oldest points are deleted.

Returns The channel number these settings were applied to or a negative error code.

**See also:**

XY trend plot dialog documentation, `CursorActive()`, `MeasureToXY()`, `MeasureX()`, `MeasureY()`

## MeasureToChan()

This creates a new Event, Marker or RealMark channel with an associated measurement process and cursor 0 iteration method for the current time view. The `Process()` command adds items to the new channel based on active cursor measurements. Before calling this function, use `MeasureX()` to set the measurement method to 102 (Time) and set `expr1$` to generate the time stamps of the new items. If you select any other method this command will return an error code. The iteration must produce times in ascending order unless the output is to a memory channel.

If you are creating a RealMark channel, you must call `MeasureY()` before this function to define the measurement to attach to each data item added to the new channel.

```
Func MeasureToChan(dest%, name$, type%, mode%, chan%, min|exp${, lv|lv${, hw{, flgs${, qu${, width{, lv2|lv2$}}}}});
```

`dest%` This is the output channel number or zero for the lowest numbered, unused memory channel. Set 1 to 400 for channels in the data file and 401 to 700 for specific memory channels. It is a fatal error to set a channel that is in use.

`name$` The output channel name. Channel units, if required, are inferred from `chan%` and the measurement method.

`type%` This sets the output channel type, as for `ChanKind()`. The allowed types are: 2 or 3 for Event, 5 for a Marker, and 7 for RealMark. If you wish to store data other than event times in the file, you should set `type%` to 7.

`mode%` This is the cursor 0 iteration mode. Modes are the same as in `CursorActive()` but not all modes can be used. Valid modes are:

|  |                    |                     |                  |                   |
|--|--------------------|---------------------|------------------|-------------------|
|  | 4 Peak find        | 8 Falling threshold | 14 Data points   | 19 Outside levels |
|  | 5 Trough find      | 12 Slope peak       | 16 Expression    | 20 Inside levels  |
|  | 7 Rising threshold | 13 Slope trough     | 17 Turning point |                   |

`chan%` This is the channel searched by the cursor 0 iterator. In expression mode (16), this is ignored and should be set to 0.

`min` This is the minimum allowed step for cursor 0 used in all modes except 16.

`exp$` This is the expression that is evaluated in mode 16.

`lv` This number or string expression sets the threshold level for threshold modes and the peak size for peak and trough modes. It is in the y axis units of channel `chan%` or y axis units per second for modes 12 and 13. In mode 14 it sets the points as a number and defaults to 1. This argument is ignored and should be 0 or omitted for modes that do not require it.

`hw` This sets the hysteresis in y axis units for threshold modes. This argument is ignored and should be 0 or omitted for modes that do not require it. For backwards compatibility, if `width` is omitted, this also sets the width in seconds for slope measurements. Put another way, all threshold measurement modes (7, 8, 19, 20) MUST set `width`, otherwise the result is nonsense; do NOT use this in new code.

`flgs%` This is the sum of option flags. Add 1 to force a common x axis Add 2 for user checks on the cursor positions. The default value is zero.

`qu$` This sets the qualification expression for the iteration. If left blank then all iteration positions will be used. If not blank, and it evaluates to non-zero, then the iteration is skipped.

`width` Set this value in seconds; use 0 or omit it in modes that do not require it. For slopes it sets the time over which the slope is measured. For peaks and troughs, it sets the maximum peak width (use 0 for no maximum). For threshold modes 7, 8, 19 and 20 it sets the minimum crossing time (Delay in the dialog)

`lv2` This number or string expression is used with `lv` to set the two threshold levels for cursor iteration modes 19 and 20.

Returns The function result is the number of the created channel.

#### See also:

`CursorActive()`, `MeasureChan()`, `MeasureX()`, `MeasureY()`, `Process()`

## MeasureToXY()

This creates a new XY view with a measurement process and cursor 0 iteration method for channels in the current time view. It creates one output channel with a default measurement method. Use `MeasureX()`, `MeasureY()` and `MeasureChan()` to edit the method and add channels. Use `Process()` to generate the plot. The new XY view is the current view and is invisible. Use `windowVisible(1)` to make it visible. These commands implement the functionality of the Measurements to XY view dialog.

```
Func MeasureToXY(mode%, chan%, min|exp${, lv|lv${ ,hw{, flgs%{, qu${, width{, lv2|lv2$}}}}});
```

`mode%` This is the cursor 0 iteration mode. Modes are the same as in `CursorActive()` but not all modes can be used. Valid modes are:

|                    |                     |                  |                   |
|--------------------|---------------------|------------------|-------------------|
| 4 Peak find        | 8 Falling threshold | 14 Data points   | 19 Outside levels |
| 5 Trough find      | 12 Slope peak       | 16 Expression    | 20 Inside levels  |
| 7 Rising threshold | 13 Slope trough     | 17 Turning point |                   |

`chan%` This is the channel searched by the cursor 0 iterator. In expression mode (16), this is ignored and should be set to 0.

`min` This is the minimum allowed step for cursor 0 used in all modes except 16.

`exp$` This is the expression that is evaluated in mode 16.

`lv` This number or string expression sets the threshold level for threshold modes and the peak size for peak and trough modes. It is in the y axis units of channel `chan%` or y axis units per second for modes 12 and 13. In mode 14 it sets the points as a number and defaults to 1. This argument is ignored and

should be 0 or omitted for modes that do not require it.

- hw This sets the hysteresis in y axis units for threshold modes. This argument is ignored and should be 0 or omitted for modes that do not require it. For backwards compatibility, if width is omitted, this also sets the width in seconds for slope measurements. Put another way, all threshold measurement modes (7, 8, 19, 20) MUST set width, otherwise the result is nonsense; do NOT use this in new code.
- flgs% This is the sum of option flags. Add 1 to force a common x axis. Add 2 for user checks on the cursor positions. The default value is zero.
- qu\$ This sets the qualification expression for the iteration. If left blank then all iteration positions will be used. If not blank, and it evaluates to non-zero, then the iteration is skipped.
- width Set this value in seconds; use 0 or omit it in modes that do not require it. For slopes it sets the time over which the slope is measured. For peaks and troughs, it sets the maximum peak width (use 0 for no maximum). For threshold modes 7, 8, 19 and 20 it sets the minimum crossing time (Delay in the dialog)
- lv2 This number or string expression together with lv sets the two threshold levels for cursor iteration modes 19 and 20.

Returns The function result is an XY view handle or a negative error code.

Arguments passed as strings are not evaluated until data is processed. Invalid strings generate invalid measurements and no data points in the XY view.

### Example

This generates a plot of peak values in channel 1 of the current time view. Peaks must be at least 0.1 seconds apart and the data must fall by at least 1 y axis unit after each peak.

```
var xy%; 'Handle of new xy view
xy%:=MeasureToXY(4, 1, 0.1, 1); 'Peak, chan 1, min step 0.1, amp 1
WindowVisible(1); 'Window is invisible, so show it
MeasureX(102, 0, "Cursor(0)"); 'x = Time, no channel, at cursor 0
MeasureY(100, 1, "Cursor(0)"); 'y = Value of chan 1 at cursor 0
MeasureChan(1, "Peaks", 0); 'Set the title, no point limit
Process(0.0, View(-1).MaxTime(),0,1); 'Process all the data
```

### See also:

XY trend plot dialog documentation, CursorActive(), MeasureChan(), MeasureX(), MeasureY(), Process()

## MeasureX()

MeasureX() and MeasureY() set the x and y part of a measurement. The settings are saved but have no effect until MeasureChan() or MeasureToChan() are used to change or create a channel. This command implements some of the functionality of the XY plot setting dialog. The current view must be the target of the measurements.

```
Func MeasureX(type%, chan%, expr1|coef% {,expr2 {,width}});
Func MeasureY(type%, chan%, expr1|coef% {,expr2 {,width}});
```

type% This sets the x or y measurement type. Values less than 100 match those for the ChanMeasure() command (except types 5 and 1 are identical). When using MeasureX() with MeasureToChan(), the only valid type% is 102.

- |         |                 |                      |              |                |
|---------|-----------------|----------------------|--------------|----------------|
| 1 Area  | 5 Area (scaled) | 9 Minimum            | 13 Abs max.  | 17 Mean in X   |
| 2 Mean  | 6 Curve area    | 1 Peak to Peak       | 14 Peak      | 18 SD in X     |
|         |                 | 0                    |              |                |
| 3 Slope | 7 Modulus       | 1 RMS Amplitude      | 15 Trough    | 19 Mean of abs |
|         |                 | 1                    |              |                |
| 4 Sum   | 8 Maximum       | 1 Standard deviation | 16 RMS error |                |
|         |                 | 2                    |              |                |

Values from 100 up are:

- 100 Value
- 104 0-based fit coefficient
- 108 Value product



101 Value difference    105 User entered value    109 Value less baseline  
 102 Time                106 Expression  
 103 Time difference    107 Value ratio

**chan%** This is the channel number for the measurement. For time, user entered and expression measurements it is ignored and should be set to 0.

**expr1** Either a real value or a string expression that sets the start time for measurements over a time range, the position for time (102) and value measurements and the expression used for measurement type 106.

**coef%** The zero-based coefficient number for measurement type 104.

**expr2** Either a real value or a string expression that sets the end time for measurements over a time range and the reference time for single-point measurements and differences. Set an empty string when **width** is required and this is not.

**width** This is the measurement width for value and value difference measurements. The default value is zero.

Returns The function return value is zero or a negative error code.

**See also:**

XY trend plot dialog documentation, Measurement descriptions, `CursorActive()`, `MeasureChan()`, `MeasureToChan()`, `MeasureToXY()`

## MeasureY()

This is identical to `MeasureX()` and sets the y part of a measurement for a measurement channel. The settings are saved but have no effect until `MeasureChan()` is used to change or create a channel. See the `MeasureX()` documentation for details.

```
Func MeasureY(type%, chan%, expr1$ {,expr2$ {,width}});
```

**See also:**

`MeasureX()`

## MemChan()

This function creates a new channel in memory and attaches it to the file in the current time view. You can have up to 300 memory channels per file. There are two variants: the first lets you specify the channels settings, the second (added at version 6.03) copies settings from an existing channel.

```
Func MemChan(type%{, szLev%{, binsz{, pre%{, trace%}}});
Func MemChan(0, copy%);
```

**type%** The type of channel to create. Codes are:

|                  |                   |             |
|------------------|-------------------|-------------|
| 1 Waveform       | 4 Level (Event+-) | 7 RealMark  |
| 2 Event (Event-) | 5 Marker          | 8 TextMark  |
| 3 Event (Event+) | 6 WaveMark        | 9 Real wave |

**szLev%** For `TextMark`, `RealMark` and `WaveMark` channels it sets the number of characters, reals or waveform points to attach to each item. In level channels 0 or omitted sets the initial level as low, 1 sets high. Set 0 or omit for other channels. Setting the initial level was added at version 7.08 and had no effect before this version.

**binsz** Used for waveform and `WaveMark` data to specify the time interval between the waveform points. This is rounded to the nearest multiple of the underlying time resolution. If you set this 0 or negative, the smallest bin size possible is set.

**pre%** This must be present for `WaveMark` data to set the number of pre-trigger points.

**trace%** Optional, default 1, sets the number of `WaveMark` traces in range 1 to 4.

**copy%** A channel from which to copy settings, including the channel comment, title, units and scale factors.

The channel must exist.

Returns The new memory channel number, or 0 if there is no free channel, or a negative error code.

Channels created by the first variant have default titles, units, comments and scale factors. You can set these with `ChanTitle$()`, `ChanUnits$()`, `ChanComment$()`, `ChanScale()` and `ChanOffset()`. This example copies a channel of any type:

```
func CopyWave%(chan%)
var mc%
mc% := MemChan(0, chan%) 'Create waveform channel
if mc%>0 then 'Created OK?
 ChanComment$(mc%, "Copy of channel "+Str$(chan%));
 MemImport(mc%, chan%, 0, MaxTime()); 'Copy data
 ChanShow(mc%); 'display new channel
endif;
return mc%; 'Return the new memory channel number
end;
```

**See also:**

`ChanNew()`, `MemDeleteItem()`, `MemDeleteTime()`, `MemGetItem()`, `MemImport()`, `MemSave()`, `MemSetItem()`

## MemDeleteItem()

This function deletes one or more items from a channel created by `MemChan()`. To delete the entire channel use `ChanDelete()`.

```
Func MemDeleteItem(chan% {,index% {,num%}});
```

chan% The channel number of a channel created by `MemChan()`.

index% The ordinal index into the channel to the item to delete. The first item is numbered 1. If you specify index -1 or omit this argument, all items are deleted.

num% The number of items to delete from `index%`. Default value is 1.

Returns The number of items deleted or a negative error code.

**See also:**

`MemChan()`, `MemDeleteTime()`, `MemGetItem()`, `MemImport()`, `MemSave()`, `MemSetItem()`

## MemDeleteTime()

This function deletes one or more items from a memory channel based on a time range. If a marker filter is set, only items in the filter are deleted unless you add 4 to `mode%`.

```
Func MemDeleteTime(chan%, mode%, t1{, t2});
```

chan% The channel number of a channel created by `MemChan()`.

mode% This sets the items to delete and how to interpret the time range `t1` and `t2`:

- 0 A single item is deleted. `t1` is the item time and `t2` is a timing tolerance (0 if `t2` is omitted). The nearest item to `t1` in the time range `t1-t2` to `t1+t2` is deleted. If there are no items in the range, nothing is deleted.
- 1 Delete all items from time `t1-t2` to `t1+t2`. If `t2` is omitted, 0 is used.
- 2 Delete the first item from time `t1` to `t2`. If `t2` is omitted it is taken as `t1`.
- 3 Delete all items from time `t1` to `t2`. If `t2` is omitted, it is taken as `t1`.
- +4 If you add 4 to the mode, any marker filter for the channel is ignored.

`t1, t2` Two times, in seconds, that set the time range for items to delete.

Returns The number of items deleted, or a negative error code.

**See also:**

MemChan(), MemDeleteItem(), MemGetItem(), MemImport(), MemSave(), MemSetItem()

## MemGetItem()

This returns information about a memory channel item. The items are identified by their ordinal position in the channel, not by time, and any mask set for markers is ignored.

```
Func MemGetItem(chan% {,index% {,code%[] {,&data$|data[]}}});
Func MemGetItem(chan%, index%, &wave|&wave%|wave[]|wave%[] {,&n%});
```

**chan%** The channel number of a channel created by MemChan().

**index%** The ordinal index into the channel to the item required. The first item is numbered 1. If you omit the index, or specify index 0, the function returns the number of items in the channel.

The remaining fields are only allowed if the index is non-zero.

**code%** This is an integer array of at least 4 elements that is returned holding the channel marker codes. If there are no markers, the codes are all set to 0.

**data** This must be a string variable for TextMark data or a real array for RealMark or WaveMark data. It is returned holding the data from the item. If the data type is incorrect for the channel, it is not changed. For an array, the points returned is the smaller of the size of the array and the number of values in the item. You can use a two dimensional array to collect WaveMark data with multiple traces. The second dimension is for the traces: `var data[points%][traces%];`

**wave** This argument collects waveforms from waveform channels. If a real variable or array is passed, the waveform data is in user units. If an integer variable or array is passed, the data is a copy of the 16-bit integer data used by Spike2 to store waveforms. The maximum number of elements copied is the array size.

When an array is used, only contiguous data is returned. A gap in the data (when the interval between two points is greater than the sample interval for the channel) terminates the data transfer.

You may find that ChanData() is easier to use when you want to collect waveform (or event) data based on a time range.

**n%** If the previous argument is an array this optional argument returns the number of data items copied into the array.

**Returns** For **index%** of 0 or omitted, the function returns the number of items in the channel. If an index is given that is outside the range of items present, the function returns -1. Otherwise it returns the time of the item.

**See also:**

ChanData(), MemChan(), MemDeleteItem(), MemDeleteTime(), MemImport(), MemSave(), MemSetItem()

## MemImport()

This function imports data into a channel created by MemChan(). There are some restrictions on the type of data channel that you can import from, depending on the type of the destination channel.

| Destination | Source                 | Restrictions                                                                        |
|-------------|------------------------|-------------------------------------------------------------------------------------|
| Waveform    | Waveform<br>All others | Data copied, but must match sample interval<br>Not available, see EventToWaveform() |
| Event       | Waveform<br>All others | Can extract event times<br>Times are extracted from the channel.                    |
| Level       | Waveform<br>All others | Can extract event times<br>Times are extracted from the channel. First time in      |

|          |                                                                  |                                                                                                                                                                                                                                                                                                                  |
|----------|------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|          |                                                                  | destination is assumed to be a low to high transition.                                                                                                                                                                                                                                                           |
| Marker   | Waveform<br>Event<br>Level<br>All others                         | Extract event times, coded for peak/trough etc.<br>Marker codes all set to 0.<br>Rising edges coded as 01, falling as 00.<br>Marker codes are copied.                                                                                                                                                            |
| TextMark | Waveform<br>Event<br>Level<br>TextMark<br>All others             | Extract event times, coded for peak/trough etc.<br>Copies times, marker codes set 0, empty strings<br>As Event but rising edges code as 01, falling as 00<br>Copies all data, strings may be truncated if too long<br>Copies marker information, empty strings                                                   |
| RealMark | Waveform<br>Event<br>Level<br>RealMark<br>WaveMark<br>All others | Extract event times, coded for peak/trough & values<br>Times copied, marker codes and reals set to 0<br>As Event but rising edges code as 01, falling as 00<br>Copied, real data truncated or zero padded as needed<br>Copied, waveform to reals, padded/truncated<br>Marker portion copied, reals filled with 0 |
| WaveMark | Waveform<br>Event<br>Level<br>WaveMark<br>All others             | Special option, event channel marks waveforms<br>Copies times, marker codes set to 0, waveform set 0<br>As Event but rising edges code as 01, falling as 00<br>Copies all data, waveforms aligned on trigger point<br>Copies marker, waveform filled with zeros                                                  |

You can extract events or markers from waveform data using peak search and level crossing techniques. You can convert waveform data to WaveMark data with a special option that chops sections of waveform data out based on event times on a third channel.

### Import compatible channel

The first command variant imports data from a compatible channel. All event and marker channels can be copied to each other, but the information transferred is the lowest common denominator of the two channel types. Missing data is padded with zeros. Waveform data is compatible with itself if both channels have the same sampling rate.

```
Func MemImport(chan%, inCh%, start, end);
```

chan% The channel number of a channel created by MemChan().

inCh% The channel number to import data from.

start The start time to collect data from.

end The end time to collect data up to (and including).

### Extracting events from waveforms

The mode%, time, level and code% arguments are used when extracting events from waveform data. The level crossing modes use linear interpolation between points to find the exact time of the waveform crossing. The peak and trough modes fit cubic splines to the points around the peak or trough to estimate the time more accurately.

When extracting events to a RealMark channel, the first RealMark value is set to the peak, trough or level associated with the event. Peak and trough levels are calculated from a cubic spline through the data values to match the event time.

When extracting events to a WaveMark channel from waveform data, the time saved is the time of the start of the waveform section, not the peak/trough or level crossing time. The saved waveform data starts the number of points before each event set by the pre% parameter to MemChan(). The WaveMark time is adjusted to match the waveform. If both channels do not have the same sampling rate, the copied waveform is set to zero.

```
Func MemImport(chan%, inCh%, start, end{, mode%, time, level{, code%}});
```

mode% The mode of data extraction. The event times are based on:

- 0 Peaks in the waveform. If the destination is a marker, these events are coded as 2 unless code% is set.

- 1 Troughs in the waveform. If the destination is a marker, the events are coded 3 unless `code%` is set.
  - 2 Waveform rising through `level`. If the destination is a marker, the events are coded 4 unless `code%` is set.
  - 3 Waveform falling through `level`. If the destination is a marker, the events are coded 5 unless `code%` is set.
  - 4 Peak and trough times. Peaks are always coded as 2, troughs are always coded as 3.
  - 5 Level crossing times. Rising levels are always coded as 4, falling as 5.
- `time` The minimum time between detected events. Use this to filter noisy signals. In modes 4 and 5, the timing is between events of the same code.
- `level` In modes 0 and 1, this is the distance that the waveform must fall after a peak or rise after a trough. In modes 2 and 3 it is the level to cross to detect an event.
- `code%` If present and positive it overrides the codes based on `mode%` that are applied to the events. The low byte of `code%` sets the first marker code; the remaining marker codes are always 0.

### WaveMark from events and waveform data

The special mode to convert waveform to WaveMark data uses an extra channel to mark the waveform sections to be extracted. The waveform must have the same sampling rate as set for the WaveMark channel. The saved waveform data starts the number of points before each event set by the `pre%` parameter to `MemChan()`.

```
Func MemImport(chan%, inCh%, start, end, eCh%);
```

`eCh%` A channel holding event times to mark the waveform sections to extract. The time saved is the time of the first point in each waveform section. If the channel contains marker codes, these are copied to the memory channel.

Returns It returns the number of items added to the channel, or a negative error code.

#### See also:

`MemChan()`, `MemDeleteItem()`, `MemDeleteTime()`, `MemGetItem()`, `MemSave()`, `MemSetItem()`

## MemSave()

This writes a channel created by `MemChan()` to the data file associated with the current window, making the data permanent. The memory channel is not changed; use `ChanDelete()` to remove it.

```
Func MemSave(chan%, dest%{, type%{, query%{, bufSz%}});
```

`chan%` A channel created by the `MemChan()` function.

`dest%` The destination channel in the file. This must be in the range 1 to the maximum number of channels allowed in the data file.

`type%` The type of data to save the data as. The type selected must be compatible with the data in the memory channel. Codes are:

|                       |                  |            |             |
|-----------------------|------------------|------------|-------------|
| 0 Same type (default) | 3 Event (Event+) | 6 WaveMark | 9 Real wave |
| 1 Waveform            | 4 Level          | 7 RealMark |             |
| 2 Event (Event-)      | 5 Marker         | 8 TextMark |             |

The special code -1 means append the memory channel to an existing channel. The new data must occur after the last item in the `dest%` channel and the `dest%` channel must be of a compatible type to the memory channel.

`query%` If this is not present or zero, and the `dest%` channel is already in use, the user is queried about overwriting it. If this is non-zero, no query is made.

`bufSz%` If present and greater than 0, this sets the disk block size in bytes for waveform and RealWave data. If omitted (recommended) or zero, a default size is used.

Returns The number of items written, or a negative error code.

**See also:**

ChanDelete(), ChanWriteWave(), MemChan(), MemDeleteItem(), MemDeleteTime(), MemGetItem(), MemImport(), MemSetItem()

**MemSetItem()**

This function edits or adds an item in a channel created by MemChan(). The item is identified by its ordinal position in the channel and any mask set for markers is ignored.

```
Func MemSetItem(chan%, index%, time{, code%[] {, data$|data[]});
Func MemSetItem(chan%, index%, time, wave|wave%|wave[]|wave%[]);
```

**chan%** The channel number of a channel created by MemChan().

**index%** The index of the item to edit. The first item is number 1. An index of 0 adds a new item to the buffer at a position set by **time** (which must be positive).

**time** The item time, or -1 for no change. If **index%** is 0, you must supply a time. For a waveform channel, it sets the time of the first data point but if there is already data in the channel, **time** is adjusted by up to half the sampling interval to be compatible with the sampling interval of the channel and the existing data.

**code%** This is an integer array of at least 4 elements that hold the marker codes for the channel. If the channel does not require marker codes, this argument is ignored. If this parameter is omitted for a channel with markers, the codes are set to 0. From version 7.10, **code%** can also be an integer holding the 4 marker codes as:  $code0 + 256*(code1 + 256*(code2 + 256*code3))$ .

**data** A string for TextMark data or a real array for RealMark or WaveMark, holding the item data. If the data type is incorrect it is ignored. The number of points or characters set is the smaller of the number passed and the number expected. For RealMark and WaveMark data, if the array is too short, the extra values are unchanged when **index%** > 0 (editing) and have the value 0 if **index%** is 0.

WaveMark and waveform real values are limited to the range  $-5*scale+offs$  to  $4.99985*scale+offs$ . A two dimensional array is allowed for WaveMark data: `var data[points%][traces%];` passed in as `data[][]`.

**wave** For a waveform you can set one value, or an array. Real values are limited as described above. For integers, the lower 16-bits of the 32-bit integer are copied to the channel (values greater than 32767 or less than -32768 will overflow).

**Returns** The function returns the index at which the data was stored. If an index is given that is outside the range of items present, the function returns -1.

**Example**

This example creates an array of events at time 1, 2, 3...10 seconds. It assumes that the current view is a time view; if not the MakeEventChan() function will return -1.

```
'Create a new memory event channel hoding the times passed in an array.
'times is an array of times to add to a new memory event channel
'returns event channel or 0 if no data or -ve if an error
func MakeEventChan%(times[])
var ec%, n%, i%;
n% := Len(times[]); 'number of points to add
if n% <= 0 then return 0 endif; 'no data
ec% := MemChan(2); 'create a new memory channel
if (ec% < 0) then return -1 endif; 'failed to create
for i% := 0 to n%-1 do
 MemSetItem(ec%, 0, times[i%]);
next;
return ec%;
end;

var times[10], memChan%;
ArrConst(times, 1);ArrIntgl(times); 'create 1,2,3..10
memChan% := MakeEventChan%(times); 'make hidden event channel
ChanShow(memChan%); 'show hidden channel
```

**See also:**

ChanWriteWave(), MemChan(), MemDeleteItem(), MemDeleteTime(), MemGetItem(), MemImport(), MemSave()

## Message()

This function displays a message in a box with an OK button that the user must click to remove the message. Alternatively, the user can press the Enter key.

```
Proc Message(form$ {,arg1 {,arg2...}});
```

**form\$** A string that defines the output format as for `Print()`. If the string includes a vertical bar, the text before the vertical bar is used as the window title. There is no limit on the length of the text string you use here, but there is a limit on the space it can occupy in the Message dialog. You are allowed up to 80 dialog units wide (about 80 wide characters) and up to 40 dialog units high (at least 40 lines). To make use of this you must split the text into lines using `\n` to insert a new line. Alternatively, you can allow Spike2 to split up the lines, but this may not achieve the best possible results.

**arg1,2** The arguments used to replace `%d`, `%f` and `%s` type formats.

You can split the message into multiple lines by including `\n` in the `form$` string. Long messages are truncated.

**See also:**

Print(), Input(), Query(), DlgCreate()

## Mid\$()

This function returns a sub-string of a string.

```
Func Mid$(text$, index% {,count%});
```

**text\$** A text string.

**index%** The starting character in the string. The first character is index 1.

**count%** The maximum characters to return. If omitted, there is no limit on the number.

**Returns** The sub-string. If **index%** is larger than `Len(text$)`, the string is empty.

**See also:**

Asc(), Chr\$(), DelStr\$(), InStr(), LCase\$(), Left\$(), Len(), Print\$(), ReadStr(), Right\$(), Str\$(), UCase\$(), Val()

## Min()

This function returns the index of the minimum value in an array, or the minimum of several real and/or integer variables.

```
Func Min(arr[]|arr%[]|val1 {,val2 {,val3...}});
```

**arr** A real or integer array.

**valn** A list of real and/or integer values to scan for a minimum.

**Returns** The minimum value or array index of the minimum.

An example finding the minimum in a sub-array holding 10 items of the original data:

```
var data[70], minPos%, minVal;
...
minPos:=Min(data[40:10]); ' returns a position between 0 and 9
minVal:=data[40+minPos]; ' value of minimum
```

**See also:**

Abs(), ATan(), Cos(), Exp(), Frac(), Ln(), Log(), Max(), Minmax(), Pow(), Rand(), Round(), Sin(), Sqrt(), Tan(), Trunc(), XYRange()

## MinMax()

Minmax() finds the minimum and maximum values for result views and time view channels with a y axis, or the minimum and maximum intervals for an event or marker channel drawn as dots or lines. Min() and Max() are preferred in result views.

```
Func MinMax(chan%, start, finish, &min, &max, {,&minP{,&maxP{, mode%
{,binSz {,trig%|edge%{, as%}}}}});
```

chan% The channel number in the time or result view.

start The start position in time for a time view, in bins for a result view.

finish The end position in time for a time view, in bins for a result view.

min The minimum value is returned in this variable.

max The maximum value is returned in this variable.

minP The position of the minimum is returned in this variable.

maxP The position of the maximum is returned in this variable.

mode% If present, this sets the drawing mode in which to find the minimum and maximum. If mode% is absent or inappropriate, the display mode is used. This parameter is ignored in a result view. See the DrawMode() command for full details of all the drawing modes. The modes in a time view are:

- 0 The standard mode for the channel.
- 1 Dots mode for events. The dotSz% argument can be used.
- 2 Lines mode.
- 3 Waveform mode. This is the only mode for waveform channels.
- 4 WaveMark mode.
- 5 Rate mode. The binSz argument sets the width of each bin.
- 6 Mean frequency mode. binSz sets the time period.
- 7 Instantaneous frequency mode. dotSz% can be used.
- 8 Raster mode. trig% sets the trigger channel, dotSz% is used.

binSz This sets the width of the rate histogram bins and the smoothing period for mean frequency mode when specifying your own mode.

trig% The trigger channel for raster displays, level data raster displays are impossible.

edge% For level data event channels. It sets which edges of the level signal are used for mean frequency, instantaneous frequency and rate modes. The values are:

- 0 Use both edges (same as omitting the parameter).
- 1 Use rising edges.
- 2 Use falling edges

as% Used with instantaneous frequency mode to determine how the data is measured. 0=Default, 1=Dots, 2=Line, 3=Skyline.

Returns Zero if all was well or a negative error code.

**See also:**

ChanValue(), DrawMode(), Min(), Max(), XYRange()



## MM (Multimedia) commands

These commands give you control over multimedia windows in Spike2 and the `MMRate()` commands lets you control how often frames are saved when sampling through the S2Video application.

|                           |                                                                                                                            |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <code>MMOpen()</code>     | Open a multimedia window associated with the current time view or get the handle of an open multimedia view.               |
| <code>MMAudio()</code>    | Get the sample rate of any audio stream in the window.                                                                     |
| <code>MMVideo()</code>    | Get the size of the video image and the average frame rate.                                                                |
| <code>MMPosition()</code> | Set or get the position in the multimedia window, link the position to cursor 0 and step forwards and backwards by frames. |
| <code>MMImage()</code>    | Copy the current image to an array.                                                                                        |
| <code>MMFrame()</code>    | Get a list of frame times within a time range. This list does not include dropped frames (frames with no data).            |
| <code>MMOffset()</code>   | Get or set the time of the first frame in the file relative to the Spike2 time.                                            |
| <code>EditCopy()</code>   | Copy the current video image to the clipboard.                                                                             |
| <code>FileSaveAs()</code> | File types 13-16 only, save video image as a bitmap.                                                                       |

### See also:

The Spike2 Video recorder, `MMAudio()`, `MMFrame()`, `MMImage()`, `MMOffset()`, `MMOpen()`, `MMPosition()`, `MMRate()`, `MMVideo()`

## MMAudio()

This returns information about the audio content of the current multimedia window.

```
Func MMAudio({&rate});
```

`rate` If present, this is returned as the sample rate per channel, in Hz.

Returns The number of audio channels or 0 if no audio or not a multimedia window.

### See also:

`MMFrame()`, `MMImage()`, `MMOffset()`, `MMOpen()`, `MMPosition()`, `MMRate()`, `MMVideo()`

## MMFrame()

This command returns a list of real frame times in the current multimedia view. If you used `s2video` to capture the video and you set a low frame rate, the low rate is achieved by dropping frames from the AVI file. Dropping frames is achieved by setting no data for the frame in the AVI file. The frame is still counted (it has to be to allow the file to be timed correctly), but not saving data allows us to save a lot of disk space. This command lets you get the time positions where real frames (frames with image data) can be found.

```
Func MMFrame(tStart, tEnd{, secs[]{, &more%});
```

`tStart` The start of the time range (in seconds) to search for video frames.

`tEnd` The end of the time range to search for frames.

`secs[]` If present, a real array to be filled with frame times.

`more%` If present, set to 1 if the time range contains more frames than can fit in the array, else 0.

Returns The return value is either the number of frames returned in the `secs` array, or if `secs` is omitted, it is the number of frames in the time range.

All frames are positioned at a time given by `offset + n%/FPS` where `offset` is the time of the first frame and `FPS` is the number of frames per second. Frames may not exist for all values of `n%`.

**See also:**

MMAudio(), MMImage(), MMOffset(), MMOpen(), MMPosition(), MMRate(), MMVideo()

## MMImage()

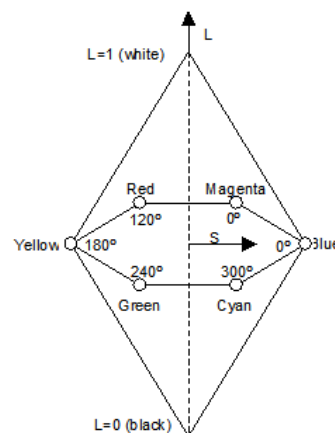
This copies the image at the current position in the current multimedia view to one or more data arrays. The two-dimensional arrays that collect the image are treated as  $x[v][h]$ , where  $v$  is a vertical co-ordinate and  $h$  is a horizontal co-ordinate.  $x[0][0]$  is at the bottom left of the image. Use `MMVideo()` to get the image size. Sections of the array that do not map onto the image are not changed.

```
Func MMImage(rgb%[][]);
Func MMImage(map[][]|mode%{, r[][]{, g[][]{, b[][]{}}});
Func MMImage(-1, h[][]{, l[][]{, s[][]{}});
```

**rgb%** If the first argument is a two-dimensional integer array, it is returned with each array element holding one pixel in RGB32 format. Bits 0-7 hold the Red intensity, bits 8-15 hold the Green and bits 16-23 hold the Blue. Bits 24-31 are 0. Any additional arguments are ignored.

**mode%** If the first argument is 1, the  $r, g$  and  $b$  arrays are filled with Red, Green and Blue intensities. If `mode%` is set to 2, the arrays are filled with Cyan, Magenta and Yellow intensities. The colour components will be in the range 0 to 1.0, inclusive. If `mode%` is 0, the  $r$  array is filled with a monochrome representation of the data with 0 representing black and 1.0 representing white; the  $g$  and  $b$  arrays are set to 0, if they are present.

If the first argument is -1, the  $h, l$  and  $s$  arrays return HLS (Hue, Lightness and Saturation) data. Hue is in degrees around a colour hexagon with Blue at 0, Red at 120 and Green at 240 degrees. Lightness represents the amount of white in a colour, from 0 (black) to 1.0 (white). Saturation is a measure of the purity of a colour, from 0 (gray) to 1.0, which is a pure colour.



**map** If this is present, it should be a 3x3 array that maps the red, green and blue (RGB) in the image into the  $r, g$  and  $b$  arrays that you provide. The first array index sets the array: 0 for  $r$ , 1 for  $g$  and 2 for  $b$ . The second array index sets the colour: 0 for red, 1 for green and 2 for blue. The array values set how much of each colour should be added into the result.

If  $R, G$  and  $B$  (all in the range 0.0 to 1.0), represent the colour of a pixel, the  $r$  array pixel is set to  $R*map[0][0]+G[*]map[0][1]+B*map[0][2]$ . The  $g$  and  $b$  arrays (if present) are calculated with the first index of `map` set to 1 and 2.

**r, g, b** These arrays, when present are filled with RGB colour information. If `map%` is used, the values in the arrays depend on the `map` values. If `mode%` is used, the values will be in the range 0 to 1.

**h, l, s** These arrays are used when the first argument is -1. They are returned holding the hue (0 to 360), lightness (0 to 1.0) and saturation (0 to 1.0).

Returns 1 if the copy succeeded and 0 if it failed.

**See also:**

EditCopy(), MMAudio(), MMFrame(), MMOffset(), MMOpen(), MMPosition(), MMRate(), MMVideo()

## MMOffset()

You can set the start time of the AVI file in the associated time view. We attempt to store this offset in the AVI file header in a supposedly unused region so that you only need do this once per file. Setting a 0 offset restores the AVI header to a standard state. You can also set an offset in the S2Video program if your camera usually produces the same offset each time you use it.

```
Func MMOffset({offset});
```

`offset` If present, this sets the new offset in seconds. The offset is stored internally to millisecond resolution. The offset is saved to the AVI file when the multimedia window closes as long as the file is not read-only.

Returns The original multimedia offset at the time of the call, in seconds.

This functions is primarily intended to allow recordings made with S2Video to be aligned. However, as we do not set a limit to the range of the offset, it can be used to align any AVI file to any offset in a file. If you do this, you will see the first or last video frame for time ranges beyond the time range spanned by the AVI file.

**See also:**

`MMAudio()`, `MMFrame()`, `MMImage()`, `MMOpen()`, `MMPosition()`, `MMRate()`, `MMVideo()`

## MMOpen()

This counts, opens and returns the view handles of multimedia windows associated with the current time or multimedia view. Multimedia files associated with `fname.smr` are named `fname-1.avi`, `fName-2.avi` and so on. If a multimedia window is opened or located, it becomes the current view. If multiple windows open, the last-opened window becomes the current view. This does not access the `s2video` application window.

```
Func MMOpen({n%{, mode%}});
```

`n%` Set to 0 to open or count all associated multimedia files. Set to `n` (greater than 0) for the `n` th multimedia file. It is not an error to open a file that is already open.

`mode%` This optional argument has the default value 0 and is the sum of the following:

- 1 Make new windows visible. Omitting this does not hide an existing window.
- 2 Reserved.
- 4 Do not open any new windows, do not change the current view. Use this to find a view handle or count the multimedia windows.

Returns If `n%` is 0 or omitted, the return value is the number of associated multimedia files that are open. If `n%` is greater than 0, and a file was opened, or was already open, the return value is the view handle.

**See also:**

`FileClose()`, `FileSaveAs()` `MMAudio()`, `MMFrame()`, `MMImage()`, `MMAOffset()`, `MMPosition()`, `MMRate()`, `MMVideo()`

## MMPosition()

This sets and gets the play position and state of the current multimedia window or sets the state of all multimedia windows associated with the current time view. Multimedia windows track any change in the associated time view position. Use this command to change the play position or state independently of the time view. Use `Rerun()` to replay a time view and all associated multimedia windows together. Multimedia windows play in real time or at the rate set by the last `Rerun()` to the window.

```
Func MMPosition({pos{, sPlay%{, &gPlay%}}});
```

`pos` If `sPlay%` is less than 2 or omitted, `pos` is the time, in seconds in the multimedia stream to move to. Omit `pos` or set it negative for no position change. If `sPlay%` is 3, `pos > 0.0` moves to the next video frame, `pos < 0.0` moves to the previous frame and `pos=0` moves to the current frame.

`sPlay%` This argument controls the play state. Omit it or set it to -1 for no change in the play state. Set 0 (or 2 or 3) to stop playing, 1 to start playing, 2 to track cursor 0 in the associated time view and to 3 to move by frames as set by `pos`.

`gPlay%` If this argument is present it is returned holding the play state after the command has run as 0 for not playing and 1 for playing.

Returns The return value is the play position after any changes made by the command have been made or -1 if an error occurred or no window was found. If the command is applied to a time view, the return value is for the lowest numbered multimedia window found.

## Move by frame

To move by frame we have to decode information in the AVI file associated with the multimedia view. If this file contains missing frame, such as will be the case where `s2video` is set to sample at a slow frame rate, we skip over missing frames and move to the next real frame. Remember that this will work accurately if the frames (or missing frames) really did occur at the frame rate held in the AVI file.

### See also:

`MMAudio()`, `MMFrame()`, `MMImage()`, `MMOffset()`, `MMOpen()`, `MMRate()`, `MMVideo()`, `Rerun()`

## MMRate()

This requests any attached listener applications (usually `s2video` applications) to change the video frame rate. You cannot run faster than the camera frame rate. The rate is achieved by dropping frames from the video stream; the result may be faster or slower than you request. This command is usually used to move between a higher frame rate to record section of interest and a lower frame rate for uninteresting data and to save disk space.

```
Func MMRate(fps);
```

`fps` The desired frames per second. Set 0 for the minimum of the camera rate or the maximum frame rate set in the `s2video` application (that is, 0 sets the best rate that the `s2video` application can give you). If you set a negative frame rate, the command makes no change to the frame rate and returns the number of listener devices (normally `s2video` applications) that are registered with Spike2.

Returns The number of listener applications that Spike2 knows about (and has set the frame rate for if `fps` is greater than or equal to 0).

The normal use of this command is to modify the recorded video frame rate. However, it can also be used to detect that the `s2video` application is present and ready to record. You might want to do this if you use `ProgRun()` to start `s2video` as it can take several seconds for the application to start, locate the multimedia hardware to use and connect to Spike2.

Setting a rate of less than 10 and more than 0 is interpreted as setting a slow rate and will be saved in the `s2video` configuration as the new slow rate. It is recommended that you always use 0 to set the normal rate, even if you know the camera frame rate. This is because setting the camera rate may still cause frames to be dropped. If your camera runs at 30 FPS and you set 30.0, you could drop every other frame due to slight timing irregularities.

### See also:

The Spike2 Video recorder, `MMAudio()`, `MMFrame()`, `MMImage()`, `MMOffset()`, `MMOpen()`, `MMPosition()`, `MMVideo()`

## MMVideo()

This returns information about the video content of the current multimedia window.

```
Func MMVideo({&hPix%, &vPix%, &fps});
```

`hPix%` If present, returned as the number of horizontal pixels in the image.

`vPix%` If present, returned as the number of vertical pixels in the image.

`fps` If present, returned as the frames per second value held in the image file. This is likely to be an approximate value. To find the times of frames in the file use `MMFrame()`.

Returns 0 if no video is present or this is not a multimedia window or 1 if there is video in the multimedia window.

### See also:

`MMAudio()`, `MMFrame()`, `MMImage()`, `MMOffset()`, `MMOpen()`, `MMPosition()`, `MMRate()`

## Modified()

This command lets you get (and in some cases, set) the modified state of a view and if it is read only. Beware that clearing the modified flag for views that support this will allow you to close the view without being prompted to save changes.

```
Func Modified({what%{, new%}});
```

*what%* 0 (or omitted) to get or set the modified state, 1 to get or set the read only state.

*new%* The new state. -1 (or omitted) for no change, 0 to clear the state, 1 to set the state.

Returns The state at the time of the call, before any change.

The meaning and effect of this routine depends on the type of the current view.

### Text view

A text view is considered modified if there are undoable changes since the last save point; such changes will be interactive edits and typing. Changes made by a script do not count as modifications and are not undoable. You can use `Modified(0,0)` to make the current state the last save point without saving the file; you cannot set the modified flag with this command.

Text views (but not the Log view) can be set read only with `Modified(1,1)` and the read only state can be cleared with `Modified(1,0)`. Note that output sequence and script files open read only if they are marked read only on disk. This command does not change the read only status of the file on disk.

### Time view

A Time window counts as modified if you make a change to it that would be written to the underlying `.smr` file. `Modified(0)` reports if such a change has been made. Changes made to memory, duplicate or virtual channels do not count as modifications. Some changes are written immediately, others are buffered up and are only written when the file is closed or committed.

### Modified(0,0)

You can force the data file to commit changes held in buffers and to write and changes in the file header with `Modified(0,0)`. Committing changes does not clear the modified flag. However, this does NOT guarantee that the changes will reach the physical disk surface, only that they are written through to the operating system. This means that should Spike2 stop working, as long as the system keeps running, your files will be OK. However, if system power fails, data may still be lost. If you use `Modified(0,0)`, all save/no save decisions made by `SampleWrite()` or triggered sampling become permanent up to the last data that was marked for saving.

### Modified(0,1)

This does everything that `Modified(0,0)` does, and then asks the operating system to flush any data associated with the file through to the disk surface. BEWARE: this can be VERY SLOW (several seconds). This performs the same actions as the Sampling configuration Automation tab Flush option. Unless your data is very precious, and you can guarantee you are in a quiescent part of the experiment where having the entire system become unresponsive for several seconds is not a problem, do NOT use `Modified(0,1)`.

### Modified(1)

`Modified(1)` reports the read only state. You cannot change the read only state of a time view; it depends on the read only state of the underlying `.smr` file.

### Result and XY views

A result or XY view is modified if changed since the last save. You can set and clear the modified flag using `Modified(0, new%)`. There is currently no concept of a read only state for a result or XY views and `Modified(1)` always returns 0.

## Other view types

The command is not implemented for other view types and will return 0.

## MousePointer()

This command loads permanent mouse pointers from external files, creates new temporary mouse pointers and can delete temporary mouse pointers when they are no longer needed. Spike2 maintains a list of standard mouse pointers (see `ToolbarMouse()` for the list); any pointers added by this function are added to this list and are available for use by the mouse `Down%`, `Move%` and `Up%` routines associated with `ToolbarMouse()`.

**Func MousePointer(text\$|nDel%);**

**text\$** This is a text string that defines a new mouse pointer. This is either the path to a data file that holds a cursor or an animated cursor, ending in ".cur" or ".ani" (not case sensitive) or it is a text string that defines a monochrome mouse pointer, as described below. The function returns the number of the new mouse pointer, or 0 if it was not created. Cursors loaded from a file are permanent and exist until Spike2 closes. There is a limit on the number of cursors that Spike2 will manage (currently set to 60); this should be more than enough for any reasonable purpose.

**nDel%** The number of a mouse pointer to delete, or -1 to delete all user-defined mouse pointers. You can only delete temporary mouse pointers, and you cannot delete a mouse pointer that is currently in use.

**Returns** The number of a newly created mouse pointer or 0 if it was not created or the number of mouse pointers that were deleted.

### Text format to create a new pointer

Mouse pointers set by this function are 32 x 32 pixel images. Inside each image is the hot spot, being the position where the mouse is deemed to point. Each image pixel can be screen coloured, black, white or the inverse of the screen. In addition, you can designate a pixel to be the hot spot. This is coded in text as follows:

| Character | Screen | Black | White | Inverse |
|-----------|--------|-------|-------|---------|
| Normal    | space  | b     | w     | i       |
| Hot-spot  | h      | B     | W     | I       |

Cursors are defined by pixel by pixel, starting at the top left, moving horizontally and starting a new row every 32 characters. However, most mouse pointers are much smaller than 32 x 32 pixels, so you can stop a line early by adding a vertical bar character "|" or by a line feed character "\n". Any character that is not a space, b, B, w, W, h, i, I, | or line feed is treated as a space. You do not need to provide 32 rows; omitted rows are treated as if they were filled with spaces. If there are multiple hot-spot characters the hot-spot position is set by the last hot-spot character. If there are none, the hot-spot is at the top left.

The following example creates a small square cursor:

```
var mp% := MousePointer(
"bbbbbbb| "
"bwwwwwb| "
"bw i wb| "
"bwihwb| "
"bw i wb| "
"bwwwwwb| "
"bbbbbbb");
```

This could be written as:

```
var mp% := MousePointer("bbbbbbb|bwwwwwb|bw i wb|bwihwb|bw i wb|bwwwwwb|bbbbbbb");
```

but the first arrangement is much easier to understand.

### Loading mouse pointers from files

Cursors created with text strings are monochrome and not animated. You can also load coloured (sometimes called 3D) cursors and animated cursors from files. If you want to experiment with this, you can find suitable files in the `WINDOWS\Cursors` folder. On my machine, the following loads an animated stopwatch cursor:

```
var ani% := MousePointer("c:\\WINDOWS\\Cursors\\stopwtch.ani");
```

**See also:**

ToolbarMouse()

## MoveBy()

This gets and sets the position of the text caret. You can move the text caret in a text window relative to the current position by lines and/or a character offset and you can extend or cancel the current selection. The caret position can be read as a position in the entire document or as a line and a column. If you use this to move past the end of a line, beware that in Windows, this is usually marked by two characters (`\r`, `\n`, `CR` `LF`).

```
Func MoveBy(sel%{, char%{, line%});
```

**sel%** With **char%** present, if **sel%** is zero, all selections are cleared. If non-zero the selection is extended to the destination of the move. With **char%** omitted **sel%=0** returns the character offset, 1 the line number and 2 the column.

**char%** If **line%** is absent, the new position is obtained by adding **char%** to the current character offset in the file. You cannot move the caret beyond the existing text.

**line%** If present it specifies a line offset. The new line is the current line number plus **line%** and the new character position is the current character position in the line plus **char%**. The new line number is limited to the existing text. If the new character position is beyond the start or end of the line it is limited to the line. You can use the `Draw()` command to position a text view to start at a given line.

**Returns** It returns the new position. `MoveBy(1,0)` returns the current position without changing the selection. See **sel%** (above) to get line and column numbers. You can use `XLow()` and `XHigh()` to get the first visible line and the line past the last fully visible line.

**See also:**

`Draw()`, `EditSelectAll()`, `MoveTo()`, `Selection$()`, `XHigh()`, `XLow()`

## MoveTo()

This moves the text caret in a text window. You position the caret by lines and/or a character offset. You can extend or cancel the text selection. The first line is number 1. See `MoveBy()` to get the caret position as a line or column.

```
Func MoveTo(sel%, char%{, line%});
```

**sel%** If zero, all selections are cleared. If non-zero the selection is extended to the destination of the move and the new current position is the start of the selection.

**char%** If **line%** is absent, this sets the new position in the file. You cannot move the caret beyond the existing text. 0 places the caret at the start of the text.

**line%** If present it specifies the new line number. The new line number is limited to the existing text. **char%** sets the position in this line (and is limited to the line). You can use the `Draw()` command to position a text view to start at a given line.

**Returns** The function returns the new position in the file. You can use `XLow()` and `XHigh()` to get the first visible line and the line past the last fully visible line.

**See also:**

`Draw()`, `EditFind()`, `EditSelectAll()`, `MoveBy()`, `Selection$()`, `XLow()`, `XHigh()`

## N

### NextTime()

This finds the next item on a channel after a time. If a marker filter is in use, only data that is included in the filter is visible. It is an error to use this function in a result view.

```
Func NextTime(chan%, time{,&val|code%[]{|data[]|data%[]|&data$}});
```

**chan%** The channel number in the view to use.

**time** The time to start the search after. Items at the time are ignored. To ensure that items at time 0 are found, set the start time of the search to a negative time.

**val** Optional: for waveform channels it returns the waveform value. For event level channels, it is returned 0 if the transition is low to high, and 1 if the transition is high to low. If there is no event it returns the level at **time**; 0 for low, 1 for high.

**code%** This optional parameter is only used if the channel is a marker type. This is an array with at least four elements that is filled in with the marker codes.

**data** Gets data from RealMark and WaveMark channels. If there is insufficient data, unused entries are unchanged. Integer arrays are for WaveMark channels and return the 16-bit data that holds the waveform. If WaveMark data has multiple traces, use a two dimensional array. The trace number is the second index.

**data\$** A string returned holding the text from a TextMark channel.

**Returns** The time of the next item, -1 if there are no more items or a negative error code.

This function can be slow if run on a channel with a marker filter set and with the majority of events filtered out as Spike2 has to search for events that are in the filter.

**See also:**

ChanData(), LastTime(), MaxTime()

## O

### Optimise()

This optimises y axes over an x axis range in time, result or XY views in the same way as the Y Range dialog optimise button. You can optimise a hidden channel. In an XY view, all channels share the same y range, so optimising affects all channels. You can also use this on a spike shape window (with no arguments) to optimise the display.

```
Proc Optimise({cSpc{, start{, finish}}});
```

**cSpc** A channel specifier for the channels to optimise or -1 for all, -2 for visible or -3 for selected. If omitted, -1 is used, for all channels.

**start** The start of the region to optimise. This is in x axis units for a time or XY view and in bins for a result view. If omitted, this is the start of the window.

**finish** The end of the region to optimise. If omitted, this is the end of the window.

**See also:**

Channel specifiers, ChanOffset(), ChanScale(), YRange(), YLow(), YHigh()



## OutputReset()

This command is equivalent to the Output Reset and Application Output Reset dialogs. It allows you to specify DAC and digital output levels to be set before and after sampling. There are two command versions: the first sets values and the second returns the set values.

```
func OutputReset(flags%, dacs%[], dacv[], dig%[][, rampt{, n1401%});
func OutputReset(&dacs%[], &dacv[], &dig%[][, &rampt{, n1401%})
```

**flags%** When to apply. Sum of: 1 = at load/program start, 2 = before sampling, 4 = after sampling

**dacs%** An array of up to 8 elements. Element *n* corresponds to DAC *n*. When setting values, set each element to 1 to apply the associated DAC value, 0 to not apply the associated value. Values of 0 and 1 are returned when reading back values.

**dacv** An array of up to 8 DAC values in Volts to apply if the corresponding element of **dacs%[]** is not zero.

**dig%** An array of up to 16 elements, element *n* corresponding to digital output bit *n*. Set values as: 1=high, 0=low, -1 no change. Elements 0-7 correspond with the output sequencer DIGLOW command, elements 8-15 correspond with the DIGOUT command.

**rampt** Ramp time in seconds, default 0. This relates to a currently unimplemented feature. It will allow you to specify how long to take to ramp the DAC outputs to their final values for use in situations where a sudden DAC change could cause a problem.

**n1401%** Currently, set this to 0 to set/get values in the Output Reset dialog (sampling configuration) and 1024 for the Application Output Reset dialog (application preferences). We have plans to allow sampling with multiple 1401s. When this is enabled you will add the 1401 number to **n1401%**. If this is omitted, this value is taken as 0, meaning set the value in the current sampling configuration.

**Return** Both function versions return the **flag%** value at the time of the call.

The **dacs%**, **dacv** and **dig%** arrays can be any length without error. If longer arrays are supplied, the extra values are ignored. If shorter arrays are supplied, only the elements present are used. For the supplied values to have any effect, at least one of the **flags%** values must be set and the desired output must be enabled with **dacs%** or **dig%**.

This command was added to Spike2 at version 7.11.

## P

### PaletteGet()

This reads back the percentages of red, green and blue in a colour in the palette.

```
Proc PaletteGet(col%, &red, &green, &blue);
```

**col%** The colour index in the palette in the range 0 to 39.

**red** The percentage of red in the colour.

**green** The percentage of green in the colour.

**blue** The percentage of blue in the colour.

This function is now deprecated. You can replace `PaletteGet(col%, r,g,b)` with `ColourGet(-1, col%, r/100, g/100, b/100)`. `PaletteGet()` should only be used when compatibility with older version of Spike2 is required.

#### See also:

`Colour dialog`, `ChanColour()`, `Colour()`, `PaletteSet()`, `ViewColour()`, `ViewUseColour()`, `XYColour()`

## PaletteSet()

This sets the colour of one of the 40 palette colours. Colours 0 to 6 form a grey scale and cannot be changed. Colours are specified using the RGB (Red, Green, Blue) colour model. For example, bright blue is (0%, 0%, 100%). Bright yellow is (100%, 100%, 0%). Black is (0%, 0%, 0%) and white is (100%, 100%, 100%).

```
Proc PaletteSet(col%, red, green, blue{, solid%});
```

**col%** The colour index in the palette in the range 0 to 39. Attempting to change a fixed colour or a non-existent colour has no effect.

**red** The percentage of red in the colour.

**green** The percentage of green in the colour.

**blue** The percentage of blue in the colour.

**solid%** If present and non-zero, sets the nearest solid colour (all pixels have the same hue in a solid colour). Systems that don't need to do this ignore **solid%**.

This function is now deprecated. You can replace `PaletteSet(col%,r,g,b)` with `ColourSet(-1, col%, r/100, g/100, b/100)`. There is no equivalent of the **solid%** argument. `PaletteSet()` should only be used when compatibility with older version of Spike2 is required.

### See also:

Colour dialog, `ChanColour()`, `Colour()`, `PaletteGet()`, `ViewColour()`, `ViewUseColour()`, `XYColour()`

## PCA()

This command performs Principal Component Analysis on a matrix of data. This can take a long time if the input matrix is large.

```
Func PCA(flags%, x[][]{, w[]{, v[][]});
```

**flags%** Add the following values to control pre-processing of the input data:

- 1 Subtract the mean value of each row from each row
- 2 Normalise each row to have mean 0.0 and variance 1.0
- 4 Subtract the mean value of each column from each column
- 8 Normalise each column to have mean 0.0 and variance 1.0

You would normally pre-process the rows or the columns, not both. If you set flags for both, the rows are processed first.

**x[][]** A  $m$  rows by  $n$  columns matrix of input data that is replaced by the output data. The first array index is the rows; the second is the columns. There must be at least as many rows as columns ( $m \geq n$ ). If you have insufficient data you can use a square matrix and fill the missing rows with zeros. If you were computing the principal components of spike data, on input, each row would be a spike waveform. On output, each row holds the proportion of each of the  $n$  principal components scaled by the **w[]** array that, when added together, would best (in a least-squares error sense) represent the input data.

**w[]** This is an optional array of length at least  $n$  that is returned holding the variance of the input data that each component accounts for. The components are ordered such that  $w[i] \geq w[i+1]$ .

**v[][]** This is an optional square matrix of size  $n$  by  $n$  that is returned holding the  $n$  principal components in the rows.

Returns 0 if the function succeeded, -1 if  $m < n$ , -2 if **w** has less than  $n$  elements or **v** has less than  $n$  rows or columns.

You can find an overview of PCA in the Clustering spikes section of the manual. In the terms of that overview, **x[][]** corresponds with the X matrix on input and the U matrix on output, **w[]** is the diagonal of W and **v[][]** is the matrix V.

## PlayOffline()

This command plays an area of the current time view through the 1401 DACs or through the sound card in the computer. If you close the view during sampling, output will cease. You cannot use the 1401 for sampling and `PlayOffline()` simultaneously. You cannot use the sound card if it is already in use. If you choose the sound card, the default wave out device is used.

Unless the waveform to replay is small (total points less than 32000), it is copied to the output device in chunks, as required. This operation happens in the background, but you must make time for it by allowing `Spike2` to idle, or by checking the play position.

There are two versions of the command; the first starts output and the second reports the play position and allows you to stop the output early.

```
func PlayOffline(cSpc,dacs%,sTime,eTime{,rep%{,scale{,flag%}}});
func PlayOffline({what%{, &rep%}});
```

**cSpc** A channel specifier for up to 4 channels to be played. These can be any mix of waveform, WaveMark or RealWave channels. The channels can have different sample rates; the rate of the first channel is taken as the rate of all channels and the data for the other channels is matched to the first by linear interpolation. Gaps in the data are output as zeros.

**dacs%** This is either an integer array of 1401 DAC channel numbers (the first DAC is numbered 0), or a single integer, being the number of the first DAC channel to use. If a single integer is used, and more than 1 channel is in use, the second, third and fourth channels use DACs  $(dacs\%+1) \bmod 4$ ,  $(dacs\%+2) \bmod 4$ ,  $(dacs\%+3) \bmod 4$ . To use DACs 4 to 7 (with the Power1401), you must explicitly list the DAC numbers.

If the DAC number for the first channel is negative, output is to the system wave out device (usually a sound card). The maximum number of channels allowed depends on your system, but will usually be at least 2.

**sTime** The start time in the current time view. This, together with **eTime** defines the data region to play.

**eTime** The end time of the region in the current time view to output.

**rep%** In the setup call, this sets the number of times to play the output. If omitted, the value 1 is used. To play for as many repeats as possible use the value 0. The total number of points to play is limited to 2 to the power 32 (about 4 billion).

In the status call, this returns the number of repeats completed.

**scale** You can scale the replay rate in the range 0.25 to 4.0 (quarter speed to four times the channel rate). If you omit this argument, 1 is used, for output at the rate of the first channel.

**flag%** The sum of: 1 to position cursor 0 at the current replay position, 2 to rerun the file linked to the waveform output, 4 to broadcast cursor 0 changes and 8 (needs 4 set) to trigger active cursor searches on cursor 0 changes. The cursor is displayed if it is hidden. When the play finishes, the cursor is hidden if it was originally hidden.

**what%** Set this to -1 to stop output. Omit or set to 0 to report the position only. Both forms of the command return the position at the time of the call or -1 if there was no play in progress.

**Returns** The setup call returns 0 if all was OK, else -1 if there is a problem with the number of channels or DAC channels, -2 for an internal setup problem, -3 if the output rate is too fast, -4 if the rate is too fast to replay in chunks, -5 if memory was exhausted, -6 for a problem with the sound card, -7 for a stupid points count. The status call returns the replay position, in seconds, or -1 if there is no replay in progress, or it has finished.

### See also:

`PlayOfflineDialog`, `DlgShow()`, `Interact()`, `PlayWaveAdd()`, `ReRun()`, `Sound()`, `Speak()`, `Toolbar()`, `Yield()`

## PlayWave...()

This family of commands gives you script control over the Play Waveform feature of Spike2. This has the following features:

- You can define a number of areas (currently up to 10) in 1401 memory that are used to replay waveforms through the 1401 DACs.
- Each area has an enable, one or two key codes, a number of DAC channels, a size, a sampling rate and a speed override, a repeat count and a link to another area.
- You can define the waveforms to play as sections of an existing Spike2 data file, or set them as calculated waveforms using the script language, or they can be reserved area that you will fill dynamically with the script language.
- Areas can be linked if they have the same number of DAC channels and the same channel list. If you link areas, subsequent areas continue to play at the same rate as the original area.
- Areas can be set to start on an external trigger.
- Space is reserved for the areas when you open the Spike2 data file for sampling. The total size of the areas is limited by 1401 memory and by the need to leave sufficient 1401 memory to allow sampling to have a chance of working.
- While sampling you can find out which area is playing and how far replay has progressed.
- You can update the waveforms in the areas while sampling and replay is in progress so it is possible to play back areas that are much larger than the available memory in the 1401.

### Triggered sampling caveat

If you set an area up to play in triggered mode, the first data point of the area is transferred to the 1401 DACs so that it can be played exactly synchronously with the trigger. If you subsequently cancel that output, the act of cancelling may allow the DACs to update. If this is likely to be a problem for you it can be a good idea to make the first (and last) point of every waveform the same value (usually 0), so that you do not get unexpected glitches.

## PlayWaveAdd()

This command adds a new area to the on-line play wave list. When you create a data file, Spike2 reserves 1401 memory and transfers any stored waveform to it. The area is set to play once and is not linked to any other area. The area replay speed factor is set to 1.0 and the wave is set to non-triggered. You must use this command before you use `FileNew()` to create the sampling window.

There are three command variants. The first adds a wave from the current time view or from a Spike2 data file (equivalent to the Add to online button in the Offline waveform output dialog), the second adds a wave from a data array, and the third reserves space without setting any data. There is a limit on the number of areas (currently 10) and it is an error to try to add a new key when the maximum number of keys is already defined. You can find how many keys exist by using `PlayWaveInfo$()` to get a string holding a list of the current keys. It is not an error to replace a key when the maximum number is defined.

```
func PlayWaveAdd(key$, lb$, dac%, sT, eT, wch%{, mem% {,path$});
func PlayWaveAdd(key$, lb$, dac%, rate, data%{ }[]{ });
func PlayWaveAdd(key$, lb$, dac%, rate, size%);
```

`key$` The first character of `key$` identifies this wave and triggers the wave playing in `SampleKey()`. It is an error to use `Chr$(0)` or an empty string as a key. It is also an error to attempt to define more keys than the maximum allowed number of areas. You can define a sub-key that can also be used to play an area with `PlayWaveKey2()`.

`lb$` The label for the play wave control bar button that will play this wave, and record the character code as a keyboard marker. Labels can be up to 7 characters long. If you include `&` as a character, it will not appear on the button, but the next character will be underlined and can be used as a keyboard shortcut.

- dac%** Either a single DAC channel number or an array of DAC channel numbers. These are the outputs that will be used to play the data. The channel numbers must be in the range 0 to 3 (0 to 7 for the Power1401) and if more than one channel is specified, the channel numbers must be different.
- sT,eT** The start and end times of the data in either the current time view, or in the file identified by the `path$` variable to be used as a source of output data.
- wch%** Either a single channel, or an array of channels to use as a data source for playing. There must be one source channel for each output channel set by the `dac%` variable. The channels can be either waveform or WaveMark data. The sample rate is taken from the sampling rate of the first channel in the list. If subsequent channels in the list have different rates, data is interpolated.
- mem%** If present, and non-zero, the data is converted to a memory image and becomes independent of the data file. Otherwise, Spike2 stores the file name and extracts the data as required for sampling (so `path$` must be supplied or the current time view file must have been saved to disk).
- path\$** This optional argument sets the name of the file to extract data from. If absent, the current time view data file is used as the data source. The named file must exist on disk and hold suitable data channels.
- rate** When the data does not come from a file, this value sets the sample rate for each channel in Hz. Spike2 will get as close to this rate as it can. The Power1401 mk II/-3 and Micro1401 mk II/-3 can set sampling intervals that are a multiple of 0.1 microseconds. The Power1401 allows intervals that are multiples of 0.2 microseconds.
- data** This is either an integer or a real array. If there is more than one DAC channel to play, the array must have the same number of rows as there are DACs, that is if there are three DACs, the array must be equivalent to `var data[n][3]`; where `n` is the number of data points. If this is an integer array, the bottom 16 bits of each element is played through the DACs. If the data is in a real array, we assume that the full range of the DACs is  $\pm 5$  Volts and that the data is the required output value in Volts.
- size%** This is the number of data points per channel to reserve for this area. The data values are not specified and you must use `PlayWaveCopy()` to transfer data to the 1401 for playing after sampling has started.

**Returns** The memory bytes in the 1401 used to hold this data, or a negative error code.

This command does not transfer the data to the 1401, that happens when the start sampling command is given. Spike2 always keeps a minimum memory area for data sampling, so there is a limit to size of the waveforms that can be copied to the 1401. This size limit is not known until sampling starts. There is also a limit on the size of a waveform that can be stored in the list, which is 32,000,000 bytes (was 2,000,000 bytes before version 5.09). Note that you can change the size of an area online with `PlayWavePoints()`; you cannot make the area larger than the size specified when sampling starts.

If you wish to link areas together or set the number of times the area is to be repeated or change the area speed factor, use one of the other `PlayWave...()` commands. The `SampleClear()` command removes all stored waveforms.

To start a waveform playing from the script you can use the `SampleKey()` function with the same key as set for the area or the output sequencer `WAVEGO` instruction.

This example generates two waveforms, one a rising ramp out of DAC 0 and the other a falling ramp from DAC 1. They are played at 1000 Hz.

```
var wave%[1024][2], dacs%[2]; 'space for the waves and dac list
ArrConst(wave%[][0], 16); ArrIntgl(wave%[][0]); 'ramp up
ArrConst(wave%[][1], -16); ArrIntgl(wave%[][1]); 'ramp down
dacs%[0] := 0; dacs%[1] := 1; 'list of dacs
PlayWaveAdd("X", "Ramps2", dacs%[], 1000, wave%[][]);
```

The next bit of code will open a data file (assuming that there is a suitable sampling configuration set), start sampling and wait for sampling actually running, then set the area to play, wait for the area to have finished playing (in the simplest way) and then stop sampling.

```
var vh%;
vh% := FileNew(0, 1); 'Create a data file (should check vh%)
SampleStart(0); 'Tell it to start
while SampleStatus() <> 2 do Yield() wend; 'Wait for running
SampleKey("X"); 'Tell area X to play
while PlayWaveStatus$() = "X" do Yield() wend; 'Wait for played
SampleStop(); 'stop sampling
```

This is using the shortest and simplest code. "Real" code would check for error return values to avoid disappointment. For example, waiting while `SampleStatus()` is not 2 is not a good idea if `SampleStart()` failed.

If you want an area to play with a triggered start add the following line after the `PlayWaveAdd()` command:

```
PlayWaveTrigger("X", 1); 'Set this area to be triggered
```

In this case, the `SampleKey("X")` command will arm the output and the Trigger LED will turn on (unless you have routed the Trigger signal to the rear panel) and the output will wait for a trigger pulse on the Trigger input or rear panel.

**See also:**

`FileNew()`, `PlayWaveChans()`, `PlayWaveCopy()`, `PlayWaveCycles()`, `PlayWaveDelete()`,  
`PlayWaveEnable()`, `PlayWaveInfo$()`, `PlayWaveKey2()`, `PlayWaveLabel$()`,  
`PlayWaveLink$()`, `PlayWavePoints()`, `PlayWaveRate()`, `PlayWaveSpeed()`,  
`PlayWaveStatus$()`, `PlayWaveStop()`, `PlayWaveTrigger()`, `SampleClear()`, `SampleKey()`

## PlayWaveChans()

This function lets you read back or set the DAC channels assigned to a particular play wave area. You cannot change the DAC assignments after sampling has started.

```
func PlayWaveChans(key${, ch%[] {, set%}});
```

**key\$** The first character of the string identifies the play wave area.

**ch%** An optional integer array used to collect or set the DAC channel numbers. The array size must match the number of channels.

**set%** If omitted or 0, the `ch%` array is filled in with the DAC channels used. If non-zero, the DAC channels are changed to the list defined by the `ch%` argument.

**Returns** The command returns the number of DACs in the area or a negative error code.

**See also:**

`PlayWaveAdd()`, `PlayWaveInfo$()`, `PlayWaveLabel$()`, `PlayWaveLink$()`, `PlayWaveRate()`

## PlayWaveCopy()

This command is used to update or read back a play wave data area in the 1401 memory. This can be done at any time that `SampleStatus()` returns 0 or 2, even while a wave is playing.

```
func PlayWaveCopy(key$, data{%}[]{}{, offs%{, read%}});
```

**key\$** The first character of the string identifies the area to be updated.

**data** This is either an integer or a real array. If there is more than one DAC channel to play, the array must have the same number of columns as there are DACs, that is if there are three DACs, the array must be equivalent to `var data[n][3];` where `n` is the number of data points. If this is an integer array, the bottom 16 bits of each element is played through the DACs. If the data is in a real array, we assume that the full range of the DACs is  $\pm 5$  Volts and that the data is the required output value in Volts. The data in the array is copied to 1401 memory. It is an error for the array size to be larger than the memory area in the 1401.

**offs%** The destination offset within the data area, in data points per channel. If the size of the data is such that the copy operation would extend beyond the end of the target area, the extra data is copied to the start of the area. If `PlayWavePoints()` has been used to reduce the area size, the copy operation wraps to the start at the size set by `PlayWavePoints()`. The first offset is 0. Use `PlayWaveStatus$()` to find the next offset to be written to the DACs.

**read%** Omit or set to 0 to set the memory in the 1401. Set to 1 to read back the 1401 data into the array. This argument was added at Spike2 version 8.01.

**Returns** The function returns 0 if all went well or if you call when there is no sampling or sampling is stopping. It returns a negative error code if there is a problem or you have requested sampling to start

and it hasn't yet started.

**See also:**

PlayWaveAdd(), PlayWaveChans(), PlayWaveCycles(), PlayWaveLink\$(),  
PlayWavePoints(), PlayWaveSpeed(), PlayWaveStatus\$(), PlayWaveStop(),  
SampleClear(), SampleStatus()

## PlayWaveCycles()

This function gets and sets the number of times to play a waveform area associated with a particular key. If this is used on-line, it will also change the number of repeats for the next play of the waveform.

```
func PlayWaveCycles(key$, {new%});
```

key\$ The first character of the string identifies the play wave area.

new% If present, this sets the number of cycles to play. The value 0 sets a very large number of cycles.

Returns The number of cycles set at the time of the call.

**See also:**

PlayWaveAdd(), PlayWaveEnable(), PlayWaveInfo\$(), PlayWaveLabel\$(),  
PlayWaveLink\$(), , PlayWaveRate(), PlayWaveSpeed(), PlayWaveStatus\$(),  
PlayWaveStop()

## PlayWaveDelete()

This function deletes one or more play wave areas from the sampling configuration. If you do this while sampling is in progress it will not change the waves loaded to the 1401.

```
func PlayWaveDelete({keys$});
```

keys\$ If this is omitted, all play wave areas are deleted. If it is present, all areas with a key that is in this string are deleted. Case is significant for play areas; "abc" and "ABC" are not the same.

Returns The number of areas deleted or a negative error code.

**See also:**

PlayWaveAdd(), SampleClear()

## PlayWaveEnable()

This function reports on the enabled state of a play wave area and optionally enables and disables it. Enabled areas are setup in the 1401 when sampling starts. Changes made with this function once sampling has started will not change the waves loaded to the 1401.

```
func PlayWaveEnable(keys$, {set%});
```

key\$ The first character of the string identifies the play wave area.

set% If present and zero, the area is disabled. If non-zero, the area is enabled.

Returns The enabled state (1=enabled, 0=disabled) of the area at the time of the call, or a negative error code.

**See also:**

PlayWaveAdd(), PlayWaveInfo\$(), PlayWaveStatus\$()

## PlayWaveInfo\$()

This function returns the list of keys associated with play wave areas, or gets the type of a particular area and the name of any associated data file.

```
func PlayWaveInfo$({key$, &size%, &type%});
```

- key\$** If omitted, the function returns the list of keys associated with the play wave areas. If present, information about a specific area is returned in the remaining arguments and the function returns any file name associated with the area.
- size%** If present, this is returned as the size of the data area in points per channel. This is not changed by `PlayWavePoints()` which sets the size actually used.
- type%** This returns the type of the area as: 0= unused, 1 = data is taken from a data file, 2 = area has a memory image of data (and may also have an associated data file), 3 = area is reserved by the script but there is no associated data.
- Returns** If there is no **key\$** value, then a string is returned holding one key character for each area. If there is a key, then the function returns the name of any data file associated with the area, in which case **type%** will be returned as 1 or 2.

**See also:**

`PlayWaveAdd()`, `PlayWaveChans()`, `PlayWaveCycles()`, `PlayWaveEnable()`,  
`PlayWaveLabel$()`, `PlayWaveLink$()`, `PlayWavePoints()`, `PlayWaveRate()`,  
`PlayWaveSpeed()`, `PlayWaveStatus$()`

## PlayWaveKey2\$()

You can define a second, sub-key, that can be used to identify an area for playing by `SampleKey()` or interactively. This can be useful when you reuse an area to have different contents or change an area size and you want to use a different key code to indicate this. You cannot use this command to make a change until after sampling has started. The original key code for the area still works if you set a second key.

```
func PlayWaveKey2$(key${ , key2$});
```

- key\$** The first character of the string identifies the play wave area.
- key2\$** If present, the first character of this string sets the sub key. This can be an empty string to clear the sub key code. If **key2\$** matches an area code, or the sub key code for any area, no change is made.
- Returns** The sub-key code that was active at the time of the call or an empty string if there was no sub-key.

**See also:**

`PlayWaveAdd()`, `PlayWavePoints()`, `PlayWaveStatus$()`, `SampleKey()`, `SampleStatus()`

## PlayWaveLabel\$()

This function returns and/or changes the label associated with each play area. The label can be up to 7 characters long and is used to label the buttons that appear on the Play waveform control bar. If you include `&` as a character, it does not appear in the label and the next character is underlined and can be used as a short cut to the button when the control bar is the current window.

```
func PlayWaveLabel$(key${ , new$});
```

- key\$** The first character of the string identifies the play wave area.
- new\$** If this is present, the label for the area is changed to the string. If the string is more than 7 characters long, only the first 7 are used.
- Returns** The label at the time of the function call.

**See also:**

`PlayWaveAdd()`

## PlayWaveLink\$()

This links play wave areas together. Linked areas must have the same number of output channels and the same output channel list. The sample rate used is the sample rate for the area that is played first. You can change links during replay if `SampleStatus()` is 0 or 2. Both the area to link from and the area to link to must exist



at the time of the call.

```
func PlayWaveLink$(key${, to$});
```

**key\$** The first character of the string identifies the play wave area.

**to\$** If present, the first character of this string sets the area to link to. Use `Chr$(0)` to cancel the link from the area set by `key$`.

**Returns** The key character of the area that was linked at the time of the call or an empty string if no area was linked or there was an error.

**See also:**

`PlayWaveAdd()`, `PlayWaveStatus$()`, `SampleStatus()`

## PlayWavePoints()

This function gets and sets the size of a waveform area associated with a particular key while sampling; it is an error to use this when not sampling data. You can only set sizes up to the original size of the area. This can be useful when you are working with a 1401 with restricted memory and need to reuse areas for different waves. If you change the size of an area while it is playing, the output immediately continues from the start of the area. This command does not affect the cycle count or any other feature of the area. You can use `PlayWaveKey2$()` to associate an additional key with this area to denote a different size or area content.

```
func PlayWavePoints(key${, new%});
```

**key\$** The first character of the string identifies the play wave area.

**new%** If present, this sets the new size in points per channel. If `new%` is zero or greater than the original area size, the original size is restored. Negative values generate an error.

**Returns** The size of the area in points per channel at the time of the call.

### Implementation

Changes made by this command (and `PlayWaveKey2$()`) are not saved to the sampling configuration and are lost each time sampling stops. Each sample area has a fixed size inside the 1401 that is allocated when sampling starts. This command allows you to make the area appear smaller than the fixed size so as to match some particular waveform. If you try to use this command before sampling starts, or refer to an area that does not exist, the script will stop with a "Play wave area does not exist" error.

**See also:**

`PlayWaveAdd()`, `PlayWaveCopy()`, `PlayWaveKey2$()`, `PlayWaveStatus$()`, `PlayWaveStop()`

## PlayWaveRate()

This function gets or sets the base play rate for a play wave area. This is the standard play rate that can be changed by `PlayWaveSpeed()`. Changes to the rate made after sampling starts have no effect on the output; use `PlayWaveSpeed()` for on-line changes.

```
func PlayWaveRate(key${, new});
```

**key\$** The first character of the string identifies the play wave area.

**new** If present, this is the new play rate for the area, in samples per second. You can set a value in the range 0.01 to 250000 Hz and `Spike2` will get as close as it can with the available hardware.

**Returns** The rate for the channel at the time of the function call.

**See also:**

`PlayWaveAdd()`, `PlayWaveLink$()`, `PlayWaveSpeed()`, `SampleKey()`

## PlayWaveSpeed()

You can alter the sample rate for a play wave area by a factor of 0.25 to 4.0 with this command. Spike2 may not be able to play at the rate you request; it will set the closest rate it can. The `Hz` argument returns the achieved rate. On-line changes are allowed.

```
func PlayWaveSpeed(key${, new{, wait%{, Hz}}});
```

`key$` The first character of the string identifies the play wave area. If this area is playing, or an area that this area links to, the rate will change during playing.

`new` If present, this is the new speed factor for the area, in the range 0.25 to 4.0. Spike2 gets as close to this speed factor as it can with the available hardware.

`wait%` If present and non-zero, any on-line speed change is postponed until the end of the current cycle and will happen within a few milliseconds of the cycle end.

`Hz` If present and a sampling document is open, it returns the real replay rate in Hz.

Returns The speed factor for the area at the time of the function call or 0 if there is no area defined by the key.

### See also:

`PlayWaveAdd()`, `PlayWaveLink$()`, `PlayWaveRate()`, `SampleKey()`

## PlayWaveStatus\$()

This function returns information about waveform output during sampling.

```
func PlayWaveStatus$({&pos%{, &cyc%}});
```

`pos%` If present, this returns the next position, in terms of the number of points per channel in the area, to write to the DAC hardware. The first position is 0. The DACs hold one data point (more if your 1401 supports a DAC Silo) ready for output on the next clock tick, so `pos%` is ahead of where the DAC has got to. If you play an area with a triggered start but do not trigger it, `pos%` is not 0. This is because the DACs are already holding at least index 0 (more with a DAC Silo), ready to output when the trigger arrives. The `pos%` argument is returned as -1 and `cyc%` returned as 1 if all data has been written to the DAC Silo, but playing has not yet finished.

`cyc%` If present, this integer variable is returned holding the number of cycles left to play (including the current cycle).

Returns The key code of the area that is playing or waiting for a trigger, or an empty string if no area is playing or sampling is not active. If an area is in Triggered mode, it counts as playing when it is waiting for a trigger.

To tell if an area in triggered mode has been triggered (started to output data), you need to see that `pos%` has changed or is set to -1.

### DAC Silo

The DAC Silo is available with a Power1401-3. It is also available with a Power1401 mk II or Micro1401-3 with the most up-to-date firmware. It allows data to be written through the DACs with a much lower processor overhead.

### See also:

`PlayWaveAdd()`, `PlayWaveCycles()`, `PlayWaveLink$()`, `PlayWaveRate()`, `PlayWaveSpeed()`, `PlayWaveStop()`, `SampleKey()`

## PlayWaveStop()

This function requests that the currently playing wave is stopped, either immediately, or when the current cycle finishes.

```
func PlayWaveStop(cEnd%);
```

`cEnd%` If present and non-zero, the current cycle for the playing area will be the last cycle for that area, otherwise output will stop immediately.

Returns 1 for OK, 0 if not playing or a negative error code.

**See also:**

`PlayWaveAdd()`, `PlayWaveCycles()`, `PlayWaveStatus$()`, `SampleKey()`

## PlayWaveTrigger()

This function reports and optionally changes the trigger state of a play wave area. If an area is triggered, a play request prepares the area but output does not start until a trigger signal is received by the 1401. This is the front panel Trigger input for the Micro1401 and Power1401 unless it is routed to the rear panel by the Edit Preferences menu.

```
func PlayWaveTrigger(key$ { , set% });
```

`key$` The first character of the string identifies the play wave area.

`set%` If present this sets the triggered state. 0 = not triggered and non-zero = triggered.

Returns The trigger state (1=triggered, 0=not triggered) of the area matching the first character in `key$` at the time of the call, or a negative error code.

If you want to know if an area in triggered mode has started to play, you can use the `PlayWaveStatus$()` command.

**See also:**

`PlayWaveAdd()`, `PlayWaveEnable()`, `PlayWaveStatus$()`

## Pow()

This function raises  $x$  to the power of  $y$ . If the calculation underflows, the result is 0.

```
Func Pow(x|x[] { []... }, y);
```

`x` A real number or a real array to be raised to the power of  $y$ .

`y` The exponent. If  $x$  is negative,  $y$  must be integral.

Returns If  $x$  is an array, it returns 0 or a negative error code. If  $x$  is a number, it returns  $x$  to the power of  $y$  unless an error is detected, when the script halts.

**See also:**

`Abs()`, `ATan()`, `Cos()`, `Exp()`, `Frac()`, `Ln()`, `Log()`, `Max()`, `Min()`, `Rand()`, `Round()`, `Sin()`, `Sqrt()`, `Tan()`, `Trunc()`

## Print()

This command prints to the current view at the text caret. If the first argument is a string (not an array), it is used as format information for the remaining arguments. If the first argument is an array or not a string or if there are more arguments than format specifiers, Spike2 prints the arguments without a format specifier in a standard format and adds a new line character at the end. If you provide a format string and you require a new line at the end of the output, include `\n` at the end of the format string (it generates the sequence CR LF in the output).

```
Func Print(form$|arg0 { ,arg1 { ,arg2... } });
```

`form$` A string that specifies how to treat the following arguments. The string contains two types of characters: ordinary text that is copied to the output unchanged and format specifiers that convert the following arguments to text. Format specifiers start with `%` and end with one of the letters `d`, `x`, `c`, `s`, `f`, `e` or `g` in upper or lower case. For a `%` in the output, use `%%` in the format string.

`arg1,2` The arguments (of any type) used to replace `%c`, `%d`, `%e`, `%f`, `%g`, `%s` and `%x` type formats. The

arguments can be arrays.

Returns 0 or a negative error code. Fields that cannot be printed are filled with asterisks.

## Format specifiers

The full format specifier is: `%{flags}{width}{.precision}format`

### flags

The `flags` are optional and can be placed in any order. They are single characters that modify the format specification as follows:

- Specifies that the converted argument is left justified in the output field.
- + Valid for numbers, and specifies that positive numbers have a + sign.
- space* If the first character of a field is not a sign, a space is added.
- 0 For numbers, causes the output to be padded on the left to the field width with 0.
- # For `x` format, `0x` is prefixed to non-zero arguments. For `e`, `f` and `g` formats, the output always has a decimal point. For `g` formats, trailing zeros are not removed.

### width

If this is omitted, the output field will be as wide as is required to express the argument. If present, it is a number that sets the minimum output field width. If the output is narrower than this, the field is padded on the left (on the right if the `-` flag was used) to this width with spaces (zeros if the `0` flag was used). The maximum width for numbers is 100.

### precision

This number sets the maximum number of characters to be printed for a string, the number of digits after the decimal point for `e` and `f` formats, the number of significant figures for `g` format and the minimum number of digits for `d` format (leading zeros are added if required). It is ignored for `c` format. There is no limit to the size of a string. Numeric fields have a maximum `precision` value of 100.

### format

The format character determines how the argument is converted into text. Both upper and lower cased version of the format character can be given. If the formatting contains alphabetic characters (for example the `e` in an exponent, or hexadecimal digits `a-f`), if the formatting character is given in upper case the output becomes upper case too (`e+23` and `0x23ab` become `E+23` and `0X23AB`). The formats are:

- c** The argument is printed as a single character. If the argument is a numeric type, it is converted to an integer, then the low byte of the integer (this is equivalent to `integer mod 256`) is converted to the equivalent ASCII character. You can use this to insert control codes into the output. If the argument is a string, the first character of the string is output. The following example prints two tab characters, the first using the standard tab escape, the second with the ASCII code for tab (8):

```
Print("\t%c", 8);
```
- d** The argument must be a numeric type and is printed as a decimal integer with no decimal point. If a string is passed as an argument the field is filled with asterisks. The following prints " 23,0002":

```
Print("%4d,%4d", 23, 2.3);
```
- e** The argument must be a numeric type; otherwise the field is filled with asterisks. The argument is printed as `{-}m.dxxxxxx{+}` where the number of `d`'s is set by the precision (which defaults to 6). A precision of 0 suppresses the decimal point unless the `#` flag is used. The exponent has at least 2 digits (in some implementations of Spike2 there may always be 3 digits, others use 2 digits unless 3 are required). The following prints "2.300000e+01,2.3E+00":

```
Print("%4e,%1E", 23, 2.3);
```
- f** The argument must be a numeric type; otherwise the field is filled with asterisks. The argument is printed as `{-}mmm.ddd` with the number of `d`'s set by the precision (which defaults to 6) and the number of `m`'s set by the size of the number. A precision of 0 suppresses the decimal point unless the `#` flag is used. The following prints "+23.000000,0002.3":

```
Print("%+f,%06.1f", 23, 2.3);
```
- g** The argument must be a numeric type; otherwise the field is filled with asterisks. This uses `e` format if the exponent is less than -4 or greater than or equal to the precision, otherwise `f` format is used. Trailing zeros and a trailing decimal point are not printed unless the `#` flag is used. The following prints "2.3e-

```
06,2.300000”:
```

```
Print(“%g,%#g”, 0.0000023, 2.3);
```

- s The argument must be a string; otherwise the field is filled with asterisks.
- x The argument must be a numeric type and is printed as a hexadecimal integer with no leading 0x unless the # flag is used. The following prints “1f,0x001F”:

```
Print(“%x,%#.4X”, 31, 31);
```

### Arrays in the argument list

The `d`, `e`, `f`, `g`, `s` and `x` formats support arrays. One dimensional arrays have elements separated by commas; two dimensional arrays use commas for columns and new lines for rows. Extra new lines separate higher dimensions. If there is a format string, the matching format specifier is applied to all elements.

### Infinity and Not a Number

The floating point number format normally stores a number within the floating point range. However, the format can also store positive infinity, negative infinity and Not a Number values. These are tricky to generate inside Spike2, but you can do it by using Virtual channel expressions like `Ch(1)/Ch(2)` where `Ch(2)` holds one or more zeros. You may also get these values if you read in data from a RealWave channel. If you try to print these values in `f` or `g` format you get `#IND` or `QNAN` for a NaN (for example the result of `0.0/0.0`) or `#INF` or `-#INF` for an infinity (for example `1.0/0.0` or `-1.0/0.0`).

#### See also:

`Message()`, `ToolbarText()`, `Print$()`, `PrintLog()`

## Print\$()

This command prints formatted output into a string. The syntax is identical to the `Print()` command, but the function returns the generated output as a string.

```
Func Print$(form$|arg0 {,arg1 {,arg2...}});
```

`form$` An optional string with formatting information. See `Print()` for a description.

`arg1,2` The data to format into a string.

Returns It returns the string that is the result of the formatting operation. Fields that cannot be printed are filled with asterisks.

#### See also:

`Asc()`, `Chr$()`, `DelStr$()`, `LCase$()`, `Left$()`, `Len()`, `Mid$()`, `Print()`, `PrintLog()`, `Right$()`, `Str$()`, `UCase$()`, `Val()`

## PrintLog()

This command prints to the log window. The syntax is identical to `Print()`. The output always goes to the log views and is always placed at the end of the view contents.

```
Func PrintLog(form$|arg0 {,arg1 {,arg2...}});
```

`form$` An optional string with formatting information. See `Print()` for a description.

`arg1,2` The data to print.

Returns 0 or a negative error code. Fields that cannot be printed are filled with asterisks.

#### See also:

`Print()`, `Print$()`, `Message()`

## Process()

Processes the current view. For result and XY views, the time view to process data from must be open. The times for start and end of processing are times in the time view. Use `View(-1).Cursor(1)` to refer to time view times from result and XY views.

```
Func Process(sTime, eTime{, clear%, opt%, dest%, gate%, len, pre{, mCd %}}});
```

- sTime** The time to start processing from, in seconds. Negative times are treated as zero. Times greater than `Maxtime()` cause no processing. In triggered modes with no trigger channel, this sets the trigger time and `eTime` is ignored. In Phase histograms with no Cycle channel, this sets the start of a cycle.
- eTime** The end time for processing. In Phase histogram mode with no Cycle channel, this sets the end time of a single cycle.
- clear%** If present, and non-zero, the result view bins are cleared before the results of the analysis are added to the result view and `Sweeps()` result is reset.
- opt%** If present and non-zero, the result view is optimised after processing the data.
- dest%** Used when processing to a channel from `MeasureToChan()`. If not used it should be set to 0. This identifies the destination channel for processing. If omitted or 0, all suitable channels are processed. It is not an error if this doesn't match a destination channel, but no processing happens.
- gate%** If present, this is the channel number of an event or marker-based channel in the associated time view to use as a gate. Both `len` and `pre` must be present.
- len** The time to process for each gate event in the time `sTime` to `eTime`.
- pre** How far before each gate event to start processing. Negative values are allowed and start the processing after the trigger.
- mCd%** If present and `gate%` is a marker or derived type, it holds the marker code to use as the gate. Set this to -1 or omit it to use the current channel marker filter.
- Returns** This returns the number of items processed. This is the number of intervals considered for an INTH (even if they didn't fall in the histogram), the number of sweeps for sweep-based analysis, the number of data blocks for `SetPower()`. In the case of an error, a negative error code is returned.

### See also:

`MeasureToChan()`, `MeasureToXY()`, `ProcessAll()`, `SetAverage()`, `SetEvtCrl()`, `SetINTH()`, `SetPhase()`, `SetPSTH()`, `SetPower()`, `SetResult()`, `SetWaveCrl()`, `Sweeps()`

## ProcessAll()

This function runs all processes that use the current time view as a data source. It is as if a `Process()` command were used for each target view using the current processing settings. These are the settings you would see if you opened the Process Settings dialog for the target. If you have already used a `Process()` command on a target, this sets the current processing settings.

```
Func ProcessAll(sTime, eTime);
```

- sTime** The time to start processing from, in seconds. Negative times are treated as zero. Times greater than `Maxtime()` cause no processing. In triggered modes with no trigger channel, this sets the trigger time and `eTime` is ignored. In Phase histograms with no Cycle channel, this sets the start of a cycle.
- eTime** The end time for processing. In Phase histogram mode with no Cycle channel, this sets the end time of a single cycle.

**Returns** Zero if no errors or a negative error code.

### See also:

`MeasureToChan()`, `Process()`, `MeasureToXY()`, `SetAverage()`, `SetEvtCrl()`, `SetINTH()`, `SetPhase()`, `SetPSTH()`, `SetPower()`, `SetResult()`, `SetWaveCrl()`, `Sweeps()`

## ProcessAuto()

This is equivalent to the Process Dialog for a New file in Automatic mode. The current view must be the view where the results appear. The processing parameters set by this command are used when this view is given a chance to update.

```
Func ProcessAuto(delay, mode%, opt%, last%, leeway%, dest%}}});
```

**delay** The minimum time between updates, in seconds.

**mode%** 0=accumulate all data. 1=clear the result, process the most recent **last** seconds.

**opt%** If present and non-zero, the result view is optimised after each process.

**last** The length of time to process in mode 1, in seconds; ignored in mode 0.

**leeway** How close cursor 0 can be to the file end for XY views and `MeasureToChan()`.

**dest%** The channel for `MeasureToChan()`, ignored for other process types. If omitted or 0, all suitable channels are processed. It is not an error if this doesn't match a destination channel, but no processing happens.

Returns 0 or a negative error code.

### See also:

`MeasureToChan()`, `MeasureToXY()`, `ProcessTriggered()`, `Process()`

## ProcessTriggered()

This is equivalent to the Process Dialog used with a New file in Gated mode. The current view must be the view where the results appear. The processing parameters set by this command are used when this view is given a chance to update.

```
Func ProcessTriggered(len, pre, gate%, clr%, opt%, {mCd%, dest%}}});
```

**len** The length of data to process around each gate event, in seconds.

**pre** The pre-event time, in seconds.

**gate%** The channel holding gate events or markers.

**clr%** If present, and non-zero, the result view bins are cleared before the results of the analysis are added to the result view and `Sweeps()` result is reset.

**opt%** If present and non-zero, the result view is optimised after processing the data.

**mCd%** If present and **gate%** is a marker or derived type, it holds the marker code to use as the gate. Set this to -1 or omit it to use the current channel marker filter.

**dest%** The channel for `MeasureToChan()`, ignored for other process types. If omitted or 0, all suitable channels are processed. It is not an error if this doesn't match a destination channel, but no processing happens.

Returns 0 or a negative error code.

### See also:

`MeasureToChan()`, `MeasureToXY()`, `ProcessTriggered()`, `Process()`

## Profile()

Spike2 saves information in the `HKEY_CURRENT_USER\Software\CED\Spike2` section of the system registry. The registry is a tree of keys with values attached to each key. If you think of the registry as a filing system, the keys are folders and the values are files. Keys and values are identified by case-insensitive text strings. The `Profile()` command manipulates the keys and values within the `Spike2` section of the registry.

You can also view and edit the registry with the `regedt32` (or `regedit`) program, which is part of your system. Select Run from the start menu and type `regedt32` then click OK. Please read the `regedt32` help information before using this program. It is a very powerful tool; careless use can severely damage your system.

Do not write vast quantities of data into the registry; it is a system resource and should be treated with respect. If you must save a lot of data, write it to a text or binary file and save the file name in the registry. If you think that you may have messed up the Spike2 section of the registry, use `regedt32` to locate the `Spike2` section and delete it. The next time you run Spike2 the section will be restored; you will lose any preferences you had set.

```
Proc Profile(key${, name${, val%{, &read%}}});
Proc Profile(key${, name${, val${, &read$}}});
```

**key\$** This sets the key to use in the `Spike2` registry section. Set `key$` empty to use the `Spike2` key. You can use nested keys separated by a backslash, for example `"My bit\stuff"` to use the key `stuff` inside the key `My bit`. The key name may not start with a backslash. Remember to use two backslashes inside quote marks; a single backslash is an escape character. It is never an error to refer to a key that does not exist; the system creates missing keys for you.

**name\$** This string identifies the data in the key to read or write. If you set an empty name, this refers to the (default) data item for the key set by `key$`.

**val** This is either a string or an integer. If `read` is omitted, `val` is written to the registry. If `read` is present, `val` is returned if the registry item does not exist, that is, it provides a "default" value.

**read** If present, it must have the same type as `val`. This is a variable that is set to the value held in the registry. If `name$` is not in the registry, `read` is set to `val`.

`Profile()` can be used with 1 to 4 arguments. It has a different function in each case:

- 1 The key identified by `key$` is deleted. All sub-keys and data values attached to the key and sub-keys are also deleted. Nothing is done if `key$` is empty.
- 2 The value identified by `name$` in the key `key$` is deleted.
- 3 The value identified by `name$` in the key `key$` is set to `val%` or `val$`.
- 4 The value identified by `name$` in the key `key$` is returned in `read%` or `read$`.

The following script example collects values at the start, then saves them at the end:

```
var path$, count%;
Profile("My data", "path", "c:\\work", path$); 'get initial path
Profile("My data", "count", 0, count%); 'and initial count
... 'your script...
Profile("My data", "path", path$); 'save final value
Profile("My data", "count", count%); 'save final count
```

A common use of the `Profile()` command is where a script starts by offering a dialog in which you prompt the user to select channels and settings. Our convention is to use the name of the script as the key and the variable name as the name, then start by collecting the values used the last time the script was run, let the user adjust the values in a dialog and then save the new values for use the next time.

## Registry use by Spike2

`HKEY_CURRENT_USER\Software\CED\Spike2` holds the following keys:

### BarList

This key holds the list of scripts (`Script0`, `Script1`...) to load into the script bar when Spike starts and the list of sampling configurations (`Sample0`, `Sample1`...) to load into the sample bar. The text associated with each keyword has the format:

```
Title|Path to the .s2s or .s2c file|Comment|code
```

See the `SampleBar()` and `ScriptBar()` commands for more. The `|code` is only used by the Sample bar.

### Edit

This key holds the editor settings for scripts, output sequences and general text editing.



## PageSetup

This key holds the margins in units of 0.01 mm for printing data views, and the margins in mm and header and footer text for text-based views.

## Preferences

The values in this key are mainly set by the Edit menu preferences. If you change any Edit menu Preferences value in this key, Spike2 will mark all preference values as unset and the next time it needs to use one, it will be read from the registry. The values are all integers except the file path, which is a string:

|                             |                                                                                                                                                            |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| E3 trigger at rear          | 0=Sample and PlayWave trigger on front panel, 1=on rear                                                                                                    |
| Enhanced Metafile           | 0=Windows metafile, 1=enhanced metafile for clipboard.                                                                                                     |
| Enter debug on error        | 0=Do not enter debug, 1=enter debug                                                                                                                        |
| Event ports at rear         | 0=Events 0&1 on front panel, 1=on rear panel.                                                                                                              |
| Fast online scroll          | 0=Normal, 1=faster (inaccurate) scrolling algorithm                                                                                                        |
| Line thickness codes        | Bits 0-3 = Axis code, bits 4-7 = Data code. The codes 0-15 map onto the 16 values in the drop down list. Bit 7=1 to use lines not rectangles to draw axes. |
| Low channels at top         | 0=Standard display shows low channel at bottom, 1=at top.                                                                                                  |
| Metafile NoCompress         | 0=Compress when multiple points per x pixel, 1=no compress.                                                                                                |
| Metafile Scale              | 0-11 selects from the list of allowed scale factors.                                                                                                       |
| New file path               | New data file directory or blank for current folder.                                                                                                       |
| No Save Prompt              | 0=Prompt to save derived views, 1=no prompt.                                                                                                               |
| Save modified scripts       | 0=Do not save, 1=save                                                                                                                                      |
| Ten volt 1401               | 0=5 Volt 1401, 1=10 Volt 1401, 2=same as last 1401.                                                                                                        |
| Use colour                  | 0=Use Black and White, 1=Use Colour (from the View menu).                                                                                                  |
| Ignore resource X range     | 0=don't ignore, 1=ignore                                                                                                                                   |
| Force idle cycles           | See Edit Preferences Sheduler                                                                                                                              |
| Update interval             | See Edit Preferences Sheduler                                                                                                                              |
| Force idle time             | See Edit Preferences Sheduler                                                                                                                              |
| Use old colours             | 0=normal, 1=force old, see Edit Preferences, Compatibility                                                                                                 |
| Use old event measures      | 0=normal, 1=force old, see Edit Preferences, Compatibility                                                                                                 |
| Use old waveform measures   | 0=normal, 1=force old, see Edit Preferences, Compatibility                                                                                                 |
| No flicker-free drawing     | 0=normal, 1=force old, see Edit Preferences, Compatibility                                                                                                 |
| No Power spectrum overlap   | 0=normal, 1=force old, see Edit Preferences, Compatibility                                                                                                 |
| No tetrode data rescaling   | 0=normal, 1=force old, see Edit Preferences, Compatibility                                                                                                 |
| Text print mode             | 0=Screen colours, 1=Invert light, 2=Black on White, 3=Colour on White                                                                                      |
| Last 1401 range             | The input range of the last 1401 seen. 5000 for +- 5V, 10000 for +- 10 V.                                                                                  |
| No Y axis invert on drag    | 0=allow invert, 1=no allow                                                                                                                                 |
| Log axis default decades    | Number of decades to display in log mode in range 2-15                                                                                                     |
| No background colour check  | 0=colours should contrast with background, 1=no check                                                                                                      |
| Maximum Log lines           | Maximum lines for display in the Log view or 0 for no limit.                                                                                               |
| Global resource file name   | The name of the file, for example "Global"                                                                                                                 |
| Global resource path        | The path, see global resources                                                                                                                             |
| Global resource file search | 0=the folder that Spike2 was run from, 1=search the data file folder, then the Spike2 folder, 2=the data file folder only                                  |
| Global resource if in path  | 0 or 1, see global resources                                                                                                                               |
| Global resource if no file  | 0 or 1, see global resources                                                                                                                               |
| Global resource use         | 0 or 1, see global resources                                                                                                                               |

We may well add more items that are not in the table. If you cannot find the option you want, set the option in Spike2 and close the program, then use the regedt32 or regedit program to look at the path:

```
HKEY_CURRENT_USER\Software\CED\Spike2\Preferences
```

You should be able to locate the string used to save the option. You can then use this string to set or clear the option.

### Dockable toolbars

The sub-keys with names starting "Bars-" are used by system code to restore dockable toolbars. Either delete them all or leave them all alone; any other change is likely to crash Spike2 on startup. If Spike2 does crash on startup, deleting all these sub-keys is a good idea; it does not do any harm (you'll get default control bar positions) and may allow the program to start.

### Recent file list

This key holds the list of recently used files that appear at the bottom of the file menu.

### Recover

This key holds the information to recover data from interrupted sampling sessions.

### Settings

This is where the evaluate bar saves the last few evaluated lines. The colour palette is also saved here. The **Cluster** sub-key holds cluster dialog settings. The **ColourMap** sub-key holds sonogram colour maps. The **Find** sub-key holds find and replace dialog settings. The **Output** sub-key holds the list of recently used sequencer files for the Sequencer tab of the Sampling Configuration dialog. The **VChan** sub-key holds the list of recent Virtual channel expressions.

### Tip

The *Tip of the Day* dialog uses this key to remember the last tip position.

### Version

Spike2 uses this key to detect when a new version of the program is run for the first time.

### Win32

In Windows NT derived systems, this key holds the desired working set sizes. The Help menu About Spike2 dialog displays the current working set sizes. See the Technical Support: Frequently asked questions section for more information about the working set and error -544.

Minimum working set Minimum size in KB (units of 1024 bytes), default is 1000

Maximum working set Maximum size in KB, default is 8000 (8 MB). You can set much larger sizes, if you wish.

## ProgKill()

This function terminates a program started using `ProgRun()`. This can be dangerous, as it will terminate a program without giving it the opportunity to save data.

```
Func ProgKill(pHdl%);
```

`pHdl%` A program handle returned by `ProgRun()`.

Returns Zero or a negative error code.

#### See also:

`ProgRun()`, `ProgStatus()`

## ProgRun()

This function runs a program using command line arguments as if from a command prompt. Use `ProgStatus()` to test the program status, `ProgKill()` to terminate it. The program inherits its environment variables from Spike2, see `System$()` for details.

```
Func ProgRun(cmd$ {,code% {,xLow, yLow, xHigh, yHigh}});
```

- cmd\$** The command string as typed at a command prompt. To run shell command *x* use "cmd /c x" in Windows NT and XP, "command /c x" in Windows 9x. To run another copy of Spike2, use "sonview.exe /M {filename}". We suspect that the longest command line that will work is 259 characters (despite the Microsoft documentation for the system call we use stating that it works for up to 32767 characters)
- code%** If present, this sets the initial application window state: 0=Hidden, 1=Normal, 2=Iconised, 3=maximised. Some programs set their own window state so this may not work. The next 4 arguments set the Normal window position:
- xLow** Position of the left window edge as a percentage of the screen width.
- yLow** Position of the top window edge as a percentage of the screen height.
- xHigh** The right hand edge as a percentage of the screen width.
- yHigh** The bottom edge position as a percentage of the screen height.
- Returns** A program handle or a negative error code. ProgStatus() releases resources associated with the handle when it detects that the program has terminated.

**See also:**

FileCopy(), FileDelete(), ProgKill(), ProgStatus(), System\$()

## ProgStatus()

This function tests if a program started with ProgRun() is still running. If it is not, resources associated with the program handle are released.

```
Func ProgStatus(pHdl%);
```

**pHdl%** The program handle returned by ProgRun().

**Returns** 1=program is running, 0=terminated, resources released, handle now invalid. A negative error code (-1525) means that the handle is invalid.

**See also:**

ProgKill(), ProgRun()

## Q

### Query()

This function is used to ask the user a Yes/No question. It opens a window with a message and two buttons. The window is removed when a button is pressed.

```
Func Query(text$, {,Yes$ {,No$}});
```

**text\$** This string forms the text in the window. If the string includes a vertical bar, the text before the vertical bar is used as the window title. There is no limit on the length of the text string you use here, but there is a limit on the space it can occupy in the Query dialog. You are allowed up to 80 dialog units wide (about 80 wide characters) and up to 40 dialog units high (at least 40 lines). To make use of this you must split the text into lines using \n to insert a new line. Alternatively, you can allow Spike2 to split up the lines, but this may not achieve the best possible results.

**Yes\$** This sets the text for the first button. If this argument is omitted, "Yes" is used.

**No\$** This sets the text for the second button. If this is omitted, "No" is used.

**Returns** 1 if the user selects Yes or presses Enter, 0 if the user selects the No button.

The following example generates a 4 line query with replacements for the Yes and No buttons.

```
Query("0123456789 123456789 123456789 123456789 123456789 123456789 123456789 \n"
 "0123456789 123456789 123456789 123456789 123456789 123456789 123456789 \n")
```

```
"0123456789 123456789 123456789 123456789 123456789 123456789 123456789 \n"
"0123456789 123456789 123456789 123456789 123456789 123456789 123456789 \n",
"Oui", "Non")
```

**See also:**

Print(), Input(), Message(),DlgCreate(), DlgFont()

## R

### Rand()

This returns pseudo-random numbers with a uniform density in a set range. The values returned are  $R * scl + off$  where  $R$  is in the range 0 up to, but not including, 1. Spike2 initialises the generator with a random seed based on the time. You must set the seed for a repeatable sequence. The sequence is independent of RandExp() and RandNorm().

```
Func Rand(seed);
Func Rand({scl, off});
Func Rand(arr[]{{[]...}}, {scl, off});
```

**seed** If present, this is a seed for the generator in the range 0 to 1. If *seed* is outside this range, the fractional part of the number is used. If you use 0.0 as the seed, the generator is initialised with a seed based on the system time.

**arr** This real or integer array is filled with random numbers. If an integer array is used, the random number is truncated to an integer.

**scl** This scales the random number. If omitted, it has the value 1.

**off** This offsets the random number. If omitted it has the value 0.

**Returns** If the first argument is not an array, the return value is a random number in the range *off* up to *off* + *scl*. If a seed is given, a random number is still returned. If the first argument is an array, the return value is 0.0.

**See also:**

RandExp(), RandNorm()

### RandExp()

This function returns pseudo-random numbers with an exponential density, suitable for generating Poisson statistics. The values returned are  $R * mean + off$  where  $R$  is a random number with the density function  $p(x) = \exp(-x)$ . When you start Spike2, the generator is initialised with a random seed based on the time. For repeatable sequences, you must set a seed. The sequence is independent of Rand() and RandNorm().

```
Func RandExp(seed);
Func RandExp({mean, off});
Func RandExp(arr[]{{[]...}}, {mean, off});
```

**seed** If present, this is a seed for the generator in the range 0 to 1. If *seed* is outside this range, the fractional part of the number is used. If you use 0.0 as the seed, the generator is initialised with a seed based on the time.

**arr** This real or integer array is filled with random numbers. If an integer array is used, the random number is truncated to an integer.

**mean** This scales the random number. If omitted, it has the value 1.

**off** This offsets the random number. If omitted it has the value 0.

**Returns** If the first argument is not an array, the return value is a random number. If a seed is given, a random number is still returned. If the first argument is an array, the return value is 0.0.

The following example fills an array with event times with a mean interval  $t$ :

```
RandExp(arr[], t); 'Fill arr with event intervals
ArrIntgl(arr[]); 'convert intervals to times
```

**See also:**

Rand(), RandNorm()

## RandNorm()

This function returns pseudo-random numbers with a normal density. The values returned are  $R * scl + off$  where  $R$  is a random number with a normal probability density function  $p(x) = \exp(-x*x/2)/\sqrt{2*pi}$ ; this has a mean of 0 and a variance of 1. When you start Spike2, the generator is initialised with a random seed based on the time. For a repeatable sequence, you must set a seed. The sequence is independent of Rand() and RandExp().

```
Func RandNorm(seed);
Func RandNorm({scl, off});
Func RandNorm(arr[]{{[]...}}, scl{, off});
```

**seed** If present, this is a seed for the generator in the range 0 to 1. If *seed* is outside this range, the fractional part of the number is used. If you use 0.0 as the seed, the generator is initialised with a seed based on the time.

**arr** This real or integer array is filled with random numbers. If an integer array is used, the random number is truncated to an integer.

**scl** This scales the random number. If omitted, it has the value 1.

**off** This offsets the random number. If omitted it has the value 0.

**Returns** If the first argument is not an array, the return value is a random number. If a seed is given, a random number is still returned. If the first argument is an array, the return value is 0.0.

**See also:**

Rand(), RandExp()

## RasterAux()

This is now deprecated; use RasterSort() or RasterSymbol() instead. It is here for version 4 compatibility, but will be removed in a future release. This function returns and optionally sets some of the raster auxiliary values for a channel in the current result view.

Values 0 and 1 are used to sort the sweeps and can be selected in addition to time order by the DrawMode() command. Unset values read back as zero.

Values 2 to 5 are the times, in seconds relative to the start of the file, of four markers (circle, cross, square and triangle) to display in the sweep. The markers are drawn in the colours set for WaveMark codes 1 to 4. Markers appear if their position relative to the sweep trigger lies within the sweep. Unset values read back as a negative time.

```
Func RasterAux(chan%, sweep%, num% {,new});
```

**chan%** The channel in the result view. The channel must have raster data enabled.

**sweep%** The sweep number in the range 1 to the number of sweeps in the channel.

**num%** The auxiliary value to return and optionally change in the range 0 to 5.

**new** If present, this changes the auxiliary value for the sweep.

**Returns** The auxiliary value at the time of the call.

Replace RasterAux(*c%*, *s%*, *num%*{, *new*}) with RasterSort(*c%*, *s%*, *num%*{, *new*}) if *num%* is 0 or 1, and with RasterSymbol(*c%*, *s%*, *num%*-2{, *new*}) if *num%* is 2 to 5.

**See also:**

DrawMode(), RasterGet(), RasterSet(), RasterSort(), RasterSymbol(), SetEvtCrl(), SetPSTH(), SetResult()

## RasterGet()

This returns a sweep of raster data for a result view channel for which raster data has been enabled. Using this when the current view is not a result view causes a fatal error.

```
Func RasterGet(chan%, sweep%, &sT{, data{%}[] {, &tick{, &eT}}});
```

- chan%** The channel number in the result view. If this is the only argument, the return value is the number of sweeps in the channel.
- sweep%** The sweep number in the result view in the range 1 to the number of sweeps. If this argument is present, the return value is the number of times in the sweep.
- sT** This is returned holding the time in seconds of the start of the sweep in the original data. This is not the sweep trigger time (the time that corresponds to the x axis 0 in the result view). The trigger time is `sT-BinToX(0)`.
- data** This optional array is filled with sweep event times. A real array returns times in seconds; an integer array returns times in units of the microseconds per time used when the view was created. The number of times copied into the array is the lesser of the number of times in the sweep and the length of the array.
- tick** This optional real value is the number of seconds per time units used when times are returned as an integer array. That is, if you multiply each integer time returned in `data%[]`, you would get the time in seconds. If you created the result view using `SetResult()` it is the tick value you passed in, otherwise it is the `BinSize()` value from the associated time view.
- eT** This optional real value is returned holding the end time of the sweep, in seconds. This is usually only of interest for data created with `SetPhase()`.

**Returns** The count of sweeps in the channel, times in the sweep or a negative error code.

**See also:**

RasterSet(), RasterSort(), RasterSymbol(), SetEvtCrl(), SetPhase(), SetPSTH(), SetResult()

## RasterSet()

This function sets the raster data for a sweep for a channel of a result view. You can replace the data for an existing sweep, or add a new sweep. It is a fatal script error to call this function when the current view is not a result view.

```
Func RasterSet(chan%, sweep%, sT {, data[]|data%[] {, eT}});
```

- chan%** The channel number in the result view.
- sweep%** This is either in the range 1 to `Sweeps()` to replace the existing raster data for a sweep, or it can be 0 to add a new sweep. If you add a new sweep, the new sweep is added to all channels. The remaining channels have a sweep added that has the same start time, and no data. If your result view has multiple channels, only use a `sweep%` value of 0 for one of the channels.
- sT** This sets the time of the start of the sweep, in seconds. This is not the trigger time of the sweep (the time that corresponds to the x axis 0 in the result view). The trigger time is `sT-BinToX(0)`.
- data** This is a real or an integer array holding the events times for the sweep. The size of the array sets the number of items. If this is a real array, the times are in seconds. If this is an integer array, the times are in the underlying tick units (see `RasterGet()` and `SetResult()` for details). Times outside the time range `start to start+MaxTime()*BinSize()` are ignored.
- eT** This optional value is the end time of the sweep. If you supply this, all times in the data array are mapped into the time period `sT to eT`, regardless of the x axis scaling set for the result view. This

argument is usually only used when working with or emulating a Phase histogram.

Returns The sweep number that received the data or a negative error code.

**See also:**

`RasterGet()`, `RasterSort()`, `RasterSymbol()`, `SetEvtCrl()`, `SetPhase()`, `SetPSTH()`

## RasterSort()

Each raster sweep has 4 values that can be used to sort displayed rasters; this function returns and optionally sets these values for the current result view. The `DrawMode()` command selects between time order and one of these values.

```
Func RasterSort(chan%, sweep%, num% {,new});
```

`chan%` The channel in the result view. The channel must have raster data enabled.

`sweep%` The sweep number in the range 1 to the number of sweeps in the channel.

`num%` The sort value to return and optionally change in the range 1 to 4.

`new` If present, this changes sort value `num%` for the sweep.

Returns The sort value at the time of the call. Unset values read back as zero.

**See also:**

`DrawMode()`, `RasterGet()`, `RasterSet()`, `RasterSymbol()`, `SetEvtCrl()`, `SetPSTH()`, `SetResult()`

## RasterSymbol()

Each raster sweep has 8 values that can be displayed as symbols; this function returns and optionally sets these values for the current result view. The values are times, in seconds relative to the start of the file. The 8 markers are: circle, cross, square, up triangle, plus, diamond, down triangle, filled square. The markers are drawn in the colours set for WaveMark codes 1 to 8. Markers appear if their position relative to the sweep trigger lies within the sweep. Unset values read back as a negative time.

```
Func RasterSymbol(chan%, sweep%, num% {,new});
```

`chan%` The channel in the result view. The channel must have raster data enabled.

`sweep%` The sweep number in the range 1 to the number of sweeps in the channel.

`num%` The symbol number in the range 1 to 8 to get or set.

`new` If present, this sets the symbol time for the sweep. Set -1 to cancel the symbol.

Returns The symbol time at the time of the call. Unset symbols have a negative time.

**See also:**

`DrawMode()`, `RasterGet()`, `RasterSet()`, `SetEvtCrl()`, `SetPSTH()`, `SetResult()`, `SetResult()`

## Read()

This function reads the next line from the current text view or external text file and converts the text into variables. The read starts at the beginning of the line containing the text cursor. The text cursor moves to the start of the next line after the read.

```
Func Read({&var1 {, &var2 {,&var3 ...}}});
```

`varn` Arguments must be variables. They can be any type. One dimensional arrays are allowed. The variable type determines how to convert the string data. In a successful call, each variable matches a field in the string, and the value of the variable changes to the value found in the field. A call to `Read()` with no arguments skips a line.

Returns The function returns the number of fields in the text line that were successfully extracted and returned in variables, or a negative error code. Attempts to read past the end of the file produce the end of file error code. If the first variable is a string, a return value of 0 means that a blank line was read.

It is not an error to run out of data before all the variables have been updated. If this is a possibility you must check that the number of items returned matches the number you expected. An array of length *n* is treated as *n* individual values.

The source string is expected to hold data values as real numbers, integer numbers (decimal or hexadecimal introduced by 0x) and strings. Strings can be delimited by quote marks, for example "This is a string", or they can be just text. If a string is not delimited, it is deemed to run to the end of the source string, so no other items can follow it. You can use `ReadSetup()` to change the characters that delimit a string and also to define hard separator characters within non-delimited strings. String delimiters are not returned as part of the string.

Normally, the fields in the source string are separated by white space (tabs and spaces) and commas. Space characters are "soft" separators. You can have any number of spaces between fields. Tabs and commas are treated as "hard" separators. Two consecutive hard separators (with or without intervening soft separators), imply a blank field. You can use `ReadSetup()` to redefine the soft and hard separators. When reading a field, the following rules are followed:

1. Soft separator (space) characters are skipped over
2. If the field is a string and the next character is a delimiter, it is skipped.
3. Characters that are legal for the destination variable are extracted until a non-legal character or a separator or a required string delimiter or end of data is found. The characters read are not converted into the variable type. If an error occurs in the translation, the function returns the error. Blank fields assigned to numbers are treated as 0. Blank fields assigned to strings produce empty strings.
4. Characters are skipped until a separator character is found or end of data. If a soft separator is found, it and any further soft separators are skipped. If the next character is a hard separator it is also skipped.
5. If there are no more variables or no more data, the process stops, else back to step 1.

## Reading exact contents of text file

If you want to use `Read()` to fill a string with the line by line contents of a text file, including leading white space, you should call `ReadSetup("", "")` before using `Read()`. This cancels all soft separators, which allows leading white space (spaces and Tab characters) to be read.

## Example

The following example shows a source line, followed by a `Read()` function, then the assignment statements that would be equivalent to the `Read()`:

```
"This is text" , 2 3 4,, 4.56 Text too 3 4 5 The source line
n := Read(fred$, jim[1:2], sam, dick%, tom%, sally$, a, b, c);
```

is equivalent to:

```
n := 7;
fred$:= "This is text";
jim[1] := 2; jim[2] := 3; sam := 4; dick% := 0; tom% := 4;
sally$:= "Text too 3 4 5"
'a, b and c are not changed
```

The following script opens a text file and writes it line by line to the Log view (including leading white space on each line):

```
var fh%, line$;
fh% := FileOpen("*.txt", 8, 0, "Select a text file"); 'open a file as text
if (fh% > 0) then ' if we opened a file...
 ReadSetup("", ""); ' cancel soft separators
 while Read(line$) >= 0 do ' read while not EOF or error
 PrintLog("%s\n", line$); ' write whatever we got
 wend;
 ReadSetup(); ' restore standard separators
 FileClose(); ' we are done with the file
endif;
```



**See also:**

EditCopy(), FileOpen(), ReadSetup(), ReadStr(), Selection\$()

## ReadSetup()

This sets the separators and delimiters used by Read() and ReadStr() to convert text into numbers and strings. You can also set string delimiters and set a string separator.

```
Proc ReadSetup({hard${, soft${, sDel${, eDel${, sSep${}}}}});
```

**hard\$** The characters to use as hard separators between all fields. If this is omitted or the string is empty, the standard hard separators of comma and tab are used.

**soft\$** The characters to use as soft separators. If this is omitted, the space character is set as a soft separator. If **soft\$** is empty, no soft separators are used.

**sDel\$** The characters that delimit the start of a string. If omitted, a double quote is used. If empty, no delimiter is set. Delimiters are not returned in the string.

**eDel\$** The characters that delimit the end of a string. If omitted, a double quote is used. If empty, no delimiter is set. If **sDel\$** and **eDel\$** are the same length, only the end delimiter character that matches the start delimiter position is used. For example, to delimit strings with <text> or 'text' set **sDel\$** to "<" and **eDel\$** to ">". You can repeat a character to force different lengths.

**sSep\$** The list of hard separator characters for strings that have no start delimiter. For example, setting " | " lets you read one|two|three into three separate strings.

**See also:**

Read(), ReadStr(), Val()

## ReadStr()

This function extracts data fields from a string and converts them into variables.

```
Func ReadStr(text$, &var1 {, &var2 {, &var3...});
```

**text\$** The string used as a source of data.

**var** The arguments must all be variables. The variables can be of any type, and can be one dimensional arrays. The type of each variable determines how the function tries to extract data from the string. See Read() for details.

**Returns** The function returns the number of fields in the text string that were successfully extracted and returned in variables, or a negative error code.

It is not an error to run out of data before all the variables have been updated. If this is a possibility you must check the returned value. If an array is passed in, it is treated as though it was the number of individual values held in the array.

**See also:**

Read(), ReadSetup(), Val()

## ReRun()

This function controls the rerun of the current time view and is equivalent to the View menu ReRun command. You cannot use this on a file that is being sampled.

```
Func ReRun({run%{, sTime{, eTime{, scale}}});
```

**run%** Set 1=start rerun, 0=stop rerun or omit for no change. Negative values return: -1=sTime, -2=eTime, -3=scale, -4=the rerun time or -1 if not rerunning.

**sTime** Sets the rerun start time. If omitted, 0 is used. This is ignored unless **run%** > 0.

`eTime` Sets the rerun end time. If omitted `MaxTime()` is used. Ignored unless `run% > 0`.

`scale` Sets the rerun time scale. If omitted, 1.0 is used. A value of 2 reruns twice as fast. Values from 0.01 to 100.0 are allowed. This is ignored unless `run% > 0`.

Returns If `run%` is positive, omitted or less than -4, the command returns the state at the time of the call: 0=not rerunning, 1=rerunning, 2=rerunning linked to a play offline waveform output, -1= rerunning is not allowed. Negative values of `run%` return the rerun settings as described for `run%`.

**See also:**

View menu Rerun, `PlayOffline()`

## Right\$()

This function returns the rightmost `n` characters of a string.

```
Func Right$(text$, n);
```

`text$` A string of text.

`n` The number of characters to return.

Returns The last `n` characters of the string, or all the string if it is less than `n` characters.

**See also:**

`Asc()`, `Chr$()`, `DelStr$()`, `InStr()`, `LCase$()`, `Left$()`, `Len()`, `Mid$()`, `Print$()`, `ReadStr()`, `Str$()`, `UCase$()`, `Val()`

## Round()

Rounds a real number or an array of reals to the nearest whole number. Values exactly half-way between two integral values round away from zero.

```
Func Round(x|x[]{|[]...});
```

`x` A real number or an array of reals.

Returns If `x` is an array it returns 0. Otherwise it returns a real number with no fractional part that is the nearest to the original number.

`Round(x)` is equivalent to:

```
Func Round(x) return (x<0.0) ? Ceil(x-0.5) : Floor(x+0.5) end
```

**See also:**

`Abs()`, `ATan()`, `Cos()`, `Exp()`, `Frac()`, `Ln()`, `Log()`, `Max()`, `Min()`, `Pow()`, `Rand()`, `Sin()`, `Sqrt()`, `Tan()`, `Trunc()`

## S

### Sample...()

All the commands with names that start with `Sample...` are either relates to setting up a sampling configuration for sampling or are related to the currently sampling data file.

## SampleAbort()

This cancels sampling a file or file sequence and closes any associated data, result and cursor views without saving them. If a result view derived from the data has been saved, the saved file remains. It is equivalent to the Abort button in the sampling control panel.

```
Func SampleAbort();
```

Returns 0 if sampling was aborted, or a negative error code.

**See also:**

SampleReset(), SampleStart(), SampleStop(), SampleStatus()

## SampleAutoComment()

This gets or sets file auto-commenting state as set in the sampling configuration dialog.

```
Func SampleAutoComment({yes%});
```

**yes%** If present a non-zero value turns on automatic prompting for file comments when sampling ends and zero turns it off. If absent, no change is made.

Returns the automatic commenting flag at the time of the function call, 0=off, 1= on.

**See also:**

FileComment\$(), SampleAutoFile(), SampleAutoName\$()

## SampleAutoCommit()

This gets or sets the automatic file commit period as set in the sampling configuration dialog. There is an overhead associated with the commit operation; do not set needlessly short time periods. A period of 0 means no file commit.

```
Func SampleAutoCommit({every%});
```

**every%** If present this sets the automatic file commit interval in seconds. The value is limited to the range 0 to 1800 seconds (a maximum of 30 minutes).

Returns the automatic commit period in seconds at the time of the function call.

**See also:**

FileSave(), SampleAutoFile(), SampleWrite()

## SampleAutoFile()

This gets or sets the automatic filing state as set in the sampling configuration dialog.

```
Func SampleAutoFile({yes%});
```

**yes%** If present, a non-zero value turns on automatic data filing when sampling ends and zero turns it off. If absent, no change is made.

Returns the automatic filing state at the time of the function call, 0=off, 1= on.

**See also:**

FileSave(), FilePathSet(), SampleAutoName\$(), SampleAutoComment(), SampleRepeats()

## SampleAutoName\$()

This gets or sets the template for file auto-naming as in the sampling configuration dialog. The path is set by `FilePathSet()`. The name must be set before opening the file for sampling for the automatic name to be used.

```
Func SampleAutoName$({name$});
```

`name$` If present, this is the template for file auto-naming. An empty string turns off auto-naming. The maximum name length is 23 characters. See the sampling configuration documentation for details on the template string.

Returns The auto-naming template at the time of the function call.

### See also:

`FilePathSet()`, `SampleAutoFile()`, `SampleAutoComment()`

## SampleBar()

This gives you access to the Sample toolbar. It allows you to set a string to be associated with a button and to read back information about the existing buttons. If you call the command with no arguments it returns the number of buttons in the toolbar. Sample bar strings are saved in the system registry; see the `Profile()` command for more information.

```
Func SampleBar({n% {, &get$}});
Func SampleBar(set$);
```

`n%` If set to -1, `get$` must be omitted, all buttons are cleared and the function returns 0. When set to the number of a button (the first button is 0), `get$` is as described below. In this case, the function returns -1 if the button does not exist, 0 if it exists and is the last button, and 1 if higher-numbered buttons exist.

`set$` The string passed in should have the format described above. The function returns the new number of buttons or -1 if all buttons are already used.

Returns See the descriptions below. Negative return values indicate an error.

### Sample bar string format

The string format is a set of up to 4 fields separated by vertical bars. The following formats are acceptable (and the label, comment or code fields can be blank).

```
path
label|path
label|path|comment
label|path|comment|code
```

If no vertical bars are found, the string is presumed to be the path to the configuration file (with extension `.s2c` or `.s2cx`). If there is one vertical bar, it is presumed to separate the label for the button from the path. If there are two vertical bars, a comment is presumed and if three, a code is also presumed to exist. So the following combinations are legal:

```
path
label|path
label|path|comment
label|path|comment|code
```

The code is a number in the range 0 to 3, being the sum of the flag values 1 and 2. If omitted, it is presumed to be 0:

- 1 Immediate start. If set (1), start sampling as soon as the Sample Bar button is clicked. If not set (0), the user must click the **Start** button in the Sample control toolbar.
- 2 Write to Disk. This only has an effect if the immediate start bit is set. If set, start sampling with write to disk enabled, else start with write to disk disabled.

When the string is read back by the `SampleBar()` command or written to the system registry, trailing blank fields and the associated vertical bar are removed.

For example, the following code clears the script bar and sets two buttons:

```
SampleBar(-1); 'clear all buttons
SampleBar("Fast|C:\\Spike3\\Fast.s2c|Fast 4 channel sampling"); 'No immediate start
SampleBar("Faster|C:\\Spike3\\FastXX.s2c|Very fast sampling|1"); 'Immediate start, write c
```

**See also:**

`App()`, `Sample toolbar`, `Profile()`

## SampleBigFile()

This function returns and optionally sets the sample file type in the sampling configuration. This is equivalent to the File type field in the Sampling Configuration dialog. The function name comes about because we have taken over the function call used for setting the *Big File* flag in Spike2 version 7.

```
Func SampleBigFile({type%});
```

`type%` If present, this sets the type of file to create:

- 0 Create an old-style 32-bit data file (\*.smr) with a maximum size of 2 GB.
- 1 Create a 32-bit data file (\*.smr) with a maximum size of 1 TB.
- 2 Create a 64-bit data file (\*.smrx) with a size limited only by the operating system.

Returns The file type set in the sampling configuration at the time of the call.

**See also:**

`FileNew()`, `FileSaveAs()`, `SampleClear()`

## SampleCalibrate()

This sets or gets the waveform or WaveMark channel calibration. It changes the units, scale and offset fields only. See the Sampling Configuration dialog for a description of these fields. When used to read back information, the `unit$`, `scale` and `offset` arguments must be variables:

```
Func SampleCalibrate(chan%, units$, scale, offset{, get%});
Func SampleCalibrate(chan%, &units$, &scale, &offset, 1);
```

`chan%` The channel number of a waveform or WaveMark channel.

`units$` The new channel units (up to 5 characters) or a string variable to read the units.

`scale` The new scale factor for the channel or a real variable to read back the scale.

`offset` The new offset for the channel or a real variable to read back the offset.

`get%` Omit or set to 0 to set the calibration, set to 1 to read it back.

Returns 0 if all was well, or a negative error code if the channel is the wrong type.

**See also:**

`SampleTitle$()`, `SampleUsPerTime()`, `SampleWaveform()`, `SampleWaveMark()`

## SampleChanInfo()

This function returns channel information that is common to most channels from the current sampling configuration. It also reports the fixed channel numbers that are associated with the special channels for keyboard, text and digital markers.

```
Func SampleChanInfo(chan%{, item%{, trace%});
```

`chan%` A channel number in the sampling configuration in the range 1-100. The value 0 is reserved. Negative values return the channel numbers of special channels:

- 1 the Keyboard channel
- 2 the TextMark channel
- 3 the Digital marker channel.

**item%** If omitted, this takes the value 0, otherwise it determines the information to return. When used with **chan%** greater than 0 the information returned is:

- 0 Channel type as returned by `ChanKind()`.
- 1 Port number used by the channel (uses `trace%` for WaveMark channels, if supplied).
- 2 Expected event rate (same as waveform rate for waveform channel).
- 3 Ideal waveform rate (same as event rate for event channel). Items 2 and 3 differ for WaveMark channels.
- 4 Actual waveform rate for waveform and WaveMark channels, 0 for others.

When used with negative value of **chan%**:

- 0 Return the special channel number even if it is not used.
- 1 Return the special channel number if it is used, or 0 if it is not used

**trace%** Optional, taken as 0 if omitted. The trace index for WaveMark channels in the range 0 to the number of traces -1.

Returns The information requested or a negative error code.

**See also:**

`ChanKind()`, `SampleCalibrate()`, `SampleTextMark()`, `SampleWaveMark()`

## SampleChannels()

This function returns and optionally sets the maximum number of channels in the sampling configuration. The next time you sample, the data file will have space for this number of channels. Spike2 versions 6-8 allows up to 400 channels. Version 5 allows up to 256 but reads up to 400. Version 4 reads files with up to 100 channels. Version 3 can only read files with 32 channels. The maximum number of channels that you can sample is 100.

**Func SampleChannels({nChan%});**

**nChan%** If present, this sets the number of channels in the next data file created for sampling in the range 32-400. Numbers outside this range are set to the nearer limit. The maximum channel number for sampling is the smaller of the maximum channels in the file or 100; you can sample with a data file set for 400 channels, but the maximum channels that you can sample with a 1401 interface is 100.

Returns The number of channels set at the time of the call.

**See also:**

`FileNew()`, `FileSaveAs()`, `SampleClear()`

## SampleClear()

This either removes a channel from the sampling configuration or resets the sampling configuration to a standard state.

**Proc SampleClear({chan%});**

**chan%** Optional argument. If present, this is the channel number to remove from the sampling configuration (set to Off). If absent, the sampling configuration is reset as described above. The use of the **chan%** argument was added at version 7.00.

The standard state is:

- a data file of 32 channels (all Off except the keyboard).
- sampling mode is Continuous.
- microseconds per time is 10 and time per ADC is 10.
- No output sequence files.

- Any output waveforms for playing during sampling are cleared.
- Stop sampling when... fields are disabled and the automatic file name template is cleared.
- All sample rate optimising is disabled (equivalent to `SampleOptimise(0,0,0)`).

**See also:**

`SampleMode()`, `SampleOptimise()`,

## SampleComment\$()

This function gets and sets the comment attached to a channel in the Sampling Configuration dialog and the label and comment used by the Sample Bar (also visible in Windows Explorer when inspecting Spike2 configuration files).

```
Func SampleComment$(chan% {,new$});
```

`chan%` The channel number in the window (1 to 100) or 0 for the Sample Bar comment or -1 for the Sample Bar label.

`new$` If present, the new comment or label. If the text is too long, it is truncated.

Returns The original comment or label.

**See also:**

`SampleCalibrate()`, `SampleTitle$()`, `SampleWaveform()`, `SampleWaveMark()`

## SampleConfig\$()

This function gets information about the current sampling configuration as text.

```
Func SampleConfig$({get%});
```

`get%` What to get, taken as 0 if omitted. Values are:

- 0 This function gets the name of the file from which the current configuration was loaded or returns an empty string if the name is unknown. Sampling clears the file name as sampling can change the configuration. The file may have a `.CFG` or `.SMR` file extension.
- 1 The configuration rendered as text using spaces to align data.
- 2 The configuration rendered as text using commas to separate fields.
- 3 The configuration rendered as text using Tab characters to separate fields.

Returns The information requested by the `get%` argument.

**See also:**

`FileSaveAs()`, `FileOpen()`, `SampleChanInfo()`, `SampleSequencer$()`

## SampleDebounce()

This function gets and optionally sets the debounce period for Event-, Event+ and Digital Marker channels in the sampling configuration. It is not an error to set this value for other channel types, but has no effect. The debounce period sets the minimum separation of acceptable events; after an event, all following events that are closer than this period are ignored. There is a small time penalty for doing this, so this parameter should be left set to 0 unless it is required.

```
Func SampleDebounce(chan%{, ms});
```

`chan%` The channel number in the sampling configuration.

`ms` If present, the new value of the debounce period, in milliseconds in the range -1 to 1000. -1 means attempt to preserve simultaneous events by giving them consecutive clock ticks.

Returns The previous value of the debounce period, in milliseconds.

**See also:**

SampleDigMark(), SampleEvents()

## SampleDigMark()

This adds the digital marker channel to the sampling configuration. The title and comment of the channel can be set with SampleTitle\$() and SampleComment\$(). You can read back channel information with SampleChanInfo().

**Func SampleDigMark(rate)**

rate The expected sustained rate for digital markers on this channel in Hz.

Returns 0 if all went well, or a negative error code.

**See also:**

SampleChanInfo(), SampleClear(), SampleComment(), SampleTitle\$()

## SampleEvent()

This function sets a channel to sample event data. The title and comment of the channel can be set with SampleTitle\$() and SampleComment\$(). You can read back channel information with SampleChanInfo().

**Func SampleEvent(chan%, port%, type%, rate);**

chan% The channel number in the file to use for this data (1-100).

port% The event port number.

type% The type of the event channel: 0=Events on a falling edge (Event-), 1=Events on a rising edge (Event+), 2=Events on both edges (Level)

rate The expected maximum sustained event rate on the channel in Hz.

Returns 0 if all went well, or a negative error code.

**See also:**

SampleChanInfo(), SampleClear(), SampleComment(), SampleChanInfo(), SampleTitle\$()

## SampleHandle()

Returns view handles linked to the current sampling view (which must exist). This can be used to position, show and hide the output sequencer and sampling control panels. App() also returns control panel handles and can be used at any time, even when there is no sampling view open.

**Func SampleHandle(which%);**

which% Selects which view handle to return:

0 Sampling time view    1 Sampling control panel    2 Sequencer control panel  
3 Sample Status bar

Returns The view handle or 0 if the view does not exist or there is no sampling view open.

**See also:**

App(), View(), ViewList(), Window(), WindowVisible()



## SampleIdle()

This function is usually not needed. In previous versions of Spike2, it gave time to the sampling process that managed the data data transfer between the data acquisition device and the data file. From Spike2 version 8 the data capture always runs in a separate thread and we no longer need to give the sampling process any time. If you call this, and a time view is sampling data or rerunning, this collects the latest time for which all data is up to date and passes this on to the windows. It also checks the state of sampling/rerun and when sampling to a sequence of data files, gives an opportunity to move on to a new file.

The background idle (which runs when the system has free time) also performs this function. If a script runs continuously without giving time to the background idle, the script will call this periodically. Many of the commands that handle data in a time view also perform this function to ensure that data is up to date.

```
Proc SampleIdle();
```

This command is only documented here for completeness and may be removed in a future version of Spike2.

## SampleKey()

Adds an event to the keyboard marker channel of a sampling file as if you had typed it, including triggering the output sequencer and arbitrary waveform output. There was no return value before version 5.04.

```
Func SampleKey(key$);
```

`key$` The first character of the string is added to the keyboard marker channel.

Returns The time stamp of the added marker in seconds or -1 if no file is sampling.

### See also:

Sequencer control, Arbitrary waveform output, `PlayWaveAdd()`, `SampleAbort()`, `SampleReset()`, `SampleStart()`, `SampleStop()`, `SampleStatus()`, `SampleText()`, `SampleWrite()`

## SampleKeyMark()

This restores the keyboard marker channel to the sampling configuration. The keyboard marker channel is automatically present in new sampling configurations and this command is only needed in rare cases when a user has imported a sampling configuration from a data file with no keyboard marker channel.

```
Func SampleKeyMark({rate})
```

`rate` The expected sustained rate for keyboard markers on this channel in Hz. If omitted, 1 Hz is set.

Returns 0 if all went well, or a negative error code.

### See also:

`SampleComment$()`, `SampleDigMark()`, `SampleEvent()`, `SampleTextMark()`, `SampleTitle$()`

## SampleLimitSize()

This corresponds to *Starting and stopping: Stop at file size* in the Automation dialog.

```
Func SampleLimitSize({size});
```

`size` The size limit for the output file, in kB. A positive value sets the size and enables the limit. A negative value sets the limit to the positive value of size, but disables the limit. A zero value, or omitting the argument, means no change.

Returns The limit before the call. If the limit is disabled, the size is returned negated.

**See also:**

SampleLimitTime(), SampleMode(), SampleRepeats()

## SampleLimitTime()

This corresponds to *Starting and stopping: Stop at time* in the Automation dialog.

```
Func SampleLimitTime({time});
```

**time** The time in seconds to set as a limit. A positive time sets the limit and enables it. A negative time sets the limit to the positive time, but disables the limit. A value of zero, or omitting the argument, leaves the time limit unchanged.

**Returns** The limit before the call. If the limit is disabled, the time is returned negated.

**See also:**

SampleLimitSize(), SampleMode(), SampleRepeats()

## SampleMode()

This function sets and gets the sampling mode in the Sampling configuration dialog. To set Triggered mode you should use `SampleTrigger()` as mode 3 is maintained here for backwards compatibility with previous versions of Spike2.

```
Func SampleMode(mode%{, time, trig%|every});
```

**mode%** This argument determines the action of the command:

- 0 Returns the current mode as 1, 2, or 3 for Continuous, Timed or Triggered. Any additional arguments are ignored.
- 1 Sets Continuous recording mode.
- 2 Sets Timed recording mode. All three arguments are required.
- 3 Sets Triggered recording mode. All three arguments are required.
- 1 Returns the `For` field value, valid in Timed mode.
- 2 Returns the value of the `Every` field, valid for Timed sampling mode.
- 3 Returns the `Trigger` field value, valid in Triggered sampling mode.

**time** In modes 2 and 3 it sets the `For` field of the Sampling mode group, in seconds.

**every** In mode 2 it sets the `Every` field for timed sampling, in seconds.

**trig%** The trigger channel number for mode 3. This channel must exist in the sampling configuration and can be of any type except a waveform or WaveMark channel.

**Returns** For modes 0-3 it returns the current sampling mode as 1-3, or a negative error code. In mode -1 to -3 it returns the information described for `mode%`, above.

If you set triggered mode, triggers 2 to 4 are disabled and the command is equivalent to calling `SampleTrigger(1, trig%, -1, 0.0, time, -1)`.

**See also:**

SampleLimitSize(), SampleLimitTime(), SampleTrigger(), SampleUsPerTime()

## SampleOptimise()

This function sets and gets the sample rate optimising settings in the Resolution tab of the Sampling configuration dialog. Spike2 can match the requested sample rates for waveform and WaveMark channels by changing the microseconds per time unit, time units per ADC convert and by making waveform channels into Quick or Slow or Copy channels.

The available ADC sample rate is divided by giving one share to each WaveMark channel, one to each Quick channel and one to the entire Slow channel group. The actual rates for each waveform channel (Quick or Slow) can then be further down-sampled by taking one point in  $n$ . For example, with 4 Slow channels, the fastest rate for a Slow channel is one quarter the fastest rate for a Quick channel.

The rates are optimised each time you change any aspect of the sampling configuration that affects the sample rate. With full optimise set and slow sampling rates it can take an appreciable time for Spike2 to search all possible combinations for the best sample rates. For this reason, `SampleClear()` sets no optimise and version 3 compatibility. We recommend that you use this command after all the other sample setup commands so that you pay the time penalty for optimising once.

```
Func SampleOptimise(opt%,group%,type%{,usLo%{,usHi%,{dis%{,burst%}}}});
Func SampleOptimise(get%);
```

`opt%` This sets the optimise method. Set -1 for no change or:

- 0 No optimise of microseconds per time or time per ADC.
- 1 Partial, optimise time per ADC, no optimise of microseconds per time.
- 2 Full, optimise both time per ADC and microseconds per time.

`group%` This controls the use of Quick channels. Set -1 for no change or:

- 0 Version 3 compatible; no Quick channels and channel divides up to 65535 only.
- 1 Group channels with the same ideal rate so they get the same actual rate. Use this value unless you really need one of the others.
- 2 Optimise for the least error in rate; channels with the same ideal rate may get different actual rates.

`type%` Set the type of 1401 to optimise for or -1 for no change. Settings marked \* relate to 1401 types that are no longer supported for sampling:

- 0 Works with all supported 1401s except a 1401*plus* with an old ADC.
- 1\* 1401*plus* with an old ADC. 5 Power1401 625 9 Power1401-3
- 2 Power1401 6\* micro1401 mk I
- 3\* 1401*plus* or a micro1401 7 Power1401 mk II
- 4 Micro1401 mk II 8 Micro1401-3

`usLo%` If present, it sets the low limit, in microseconds, for optimising microseconds per time unit (when `opt%` is 2). Values outside the range 1 to 1000 are ignored.

`usHi%` If present, it sets the high limit, in microseconds, for optimising microseconds per time unit. Values outside the range 1 to 1000 are ignored.

`dis%` If present and not set to -1, this disables features for backward compatibility with versions of Spike2 before 6.05 and with older 1401s. `dis%` is the sum of:

- 1 No 10 MHz clock. Forces microseconds per time to be an integral number.
- 2 No WaveMark copy. If a waveform uses the same 1401 port as a WaveMark the Power1401 mk II would normally take only one data sample for WaveMark and then copy it for the waveform. Set this flag to prevent this.
- 4 No WaveMark divide. Set this bit to stop WaveMark channels using a divider. This stops a waveform channel sampling any faster than a WaveMark channel.
- 8 No dummy channels. When supported by the 1401, Spike2 normally adds in up to 4 dummy channels into the sampling if this improves the match between the requested sample rate and the actual rate.
- 16 No 64-bit optimisation of Microseconds per time unit in full optimise mode to give the same time units as Spike2 version 7.

`burst%` If present and not set to -1, this sets the state of the Burst mode check box. Set 1 for burst mode, 0 for non-burst mode.

`get%` Use the function with one argument to read back the current settings. The values 0 to 6 read back the current values of `opt%`, `group%`, `type%`, `usLo%`, `usHi%`, `dis%` and `burst%`.

Returns The setting version returns the current optimise method. When called with one argument, the return values are as documented for `get%`.

#### See also:

`SampleLimitSize()`, `SampleLimitTime()`, `SampleTimePerAdc()`, `SampleUsPerTime()`

## SampleRepeats()

This gets and optionally sets the Sampling configuration Automation tab Repeats field, which is used to sample a sequence of files with automatically incrementing names. You can use it online to return the number of files left to sample. You can choose to trigger each file in the sequence, or just the first with `SampleStartTrigger()`.

```
Func SampleRepeats({files%});
```

`files%` The number of files to sample; 0 and 1 both sample a single file. Set -1 to return the number of files that remain to be sampled (including the current file). If you omit this argument, the number of files is not changed.

Returns The number of repeats set at the time of the call or the number of files that remain to be sampled.

You must also set a file name with `SampleAutoName$()`, set automatic file saving with `SampleAutoFile(1)` and set a time or size limit with `SampleLimitTime()` or `SampleLimitSize()`, or load a suitable sampling configuration with these values set.

To make life easier for the script programmer, each file in the sequence has the same view handle, which is the handle of the first file you create with `FileNew()`. As each file reaches the time or size limit, it is saved, closed and replaced with the next file. The view handle always references a file, but its file name and sampling status will vary. The last sampled file in the sequence is not closed. You can use `SampleStop()` or `SampleAbort()` to end a sequence early and `SampleReset()` to restart a file.

### See also:

`SampleAutoFile()`, `SampleAutoName$()`, `SampleLimitSize()`, `SampleLimitTime()`, `SampleReset()`, `SampleStart()`, `SampleStartTrigger()`, `SampleStatus()`, `SampleStop()`

## SampleReset()

This abandons sampling, deletes any data that has been written to disk, and returns to a state as if `FileNew()` had just been used to create a new data file. If it is used when sampling a sequence of files, the current file is restarted and the sequence continues.

```
Func SampleReset();
```

Returns 0 if the reset operation completed without a problem, or a negative error code.

### See also:

`SampleAbort()`, `SampleStart()`, `SampleStop()`, `SampleStatus()`, `SampleWrite()`

## SampleSeqClock()

This command changes the output sequence tick rate. You can change the default rate used when compiling a sequence that does not contain a `SET` or `SCLK` directive. You can also dynamically change the speed of a running sequence (the sequencer tick period is not guaranteed to be accurate during the change). When you change the default tick, time periods set in a sequence by `s()`, `ms()` and `us()` are set as accurately as the tick rate allows. If you change the rate of a running sequence, all sequencer-tick related periods, including periods set by `s()`, `ms()` and `us()`, are changed.

```
Func SampleSeqClock({which%{, ms}});
```

`which%` Set to 0 (or omit) to get or set the default tick period used when compiling a sequence. Set to 1 to get or set the tick period in use by a running sequence.

`ms` If present and greater than 0, this sets the new tick period in milliseconds.

Returns The tick period in milliseconds at the time of the call. If you set `which%` to 1 when there is no running sequence, the returned value is 0.

### See also:

`SampleKey()`, `SampleSeqStep()`, `SampleSequencer$()`, `SampleStart()`

## SampleSeqCtrl()

This gets and optionally sets options that control the use of the output sequence. You can read more about these options in the documentation for the Sequencer tab of the Sampling Configuration dialog.

```
Func SampleSeqCtrl(opt%{, new%});
```

`opt%` There are three options:

- 1 The sequencer jump control. The values you can set with `new%` or read back are: 0 = keyboard, control panel and script, 1 = control panel and script, 2 = script only. It is an error to set values outside this range.
- 2 The minimum number of sequencer instructions to reserve space for. Values outside the range 0-8192 are set to the nearer limit. This is used when you load new sequences during sampling as space for the largest sequencer must be reserved before you start. Space is always reserved for the sequence that is loaded at the start of sampling, so setting a value of 0 is fine if you will not be loading any additional sequences during sampling, or if they are all the same size or smaller than the original.
- 3 The minimum number of table entries to reserve space for. Values outside the range 0-1000000 are set to the nearer limit. This is used when you are going to load new sequences during sampling. Space is always reserved for the table usage of the sequence that is loaded when sampling starts, so setting a value of 0 is fine if you will not load additional sequences, or if they all use the same or less space than the original sequence.

`new%` The new value for the control option.

Returns The value of the option selected by `opt%` at the time of the call (before it is changed by `new%`).

**See also:**

`SampleKey()`, `SampleSeqStep()`, `SampleSequencer$()`, `SampleStart()`

## SampleSeqStep()

This returns the current sequencer step or -1 if not sampling. If no sequence is running the result is usually 0 (but this is not guaranteed).

```
Func SampleSeqStep();
```

Returns The current sequence step number, or -1 if not sampling.

**See also:**

`SampleKey()`, `SampleSequencer$()`, `SampleSeqVar()`

## SampleSeqTable()

If there is a sampling document with an output sequence, you can use this function to find the size of any table set in the sequence by the `TABSZ` directive. You can also use this to transfer data between an integer array and the table

```
Func SampleSeqTable({tab%[]{, offs%{, get%}}});
```

`tab%[]` An integer array holding items to transfer to the 1401 sequencer table or to hold items read back from the table. The array size sets the maximum item count

`offs%` This sets the index into the sequencer table to start the transfer. The first index in the table is 0. If this value is negative or greater than or equal to the sequencer table size, no data is transferred. If omitted, the value 0 is used.

`get%` Set 0 or omit this argument to transfer data to the sequencer table, set to 1 to transfer data from the sequencer table.

Returns If you call this with no arguments, the return value is the size of the sequencer table. Otherwise, the returned value is the number of items transferred between the sequencer table and the array. A

negative error code is also possible, for example -1 if there is no sampling document.

A sampling document must exist before you can use this function. See `FileNew(0, ...)` to create a sampling document from the script language.

**See also:**

`SampleKey()`, `SampleSequencer$()`, `SampleSeqVar()`

## SampleSeqTick0

If there is a sampling document open and the output sequencer is running, this command lets you read and set the zero value used by the `TICKS` sequencer instruction and set by the `TICK0` instruction.

```
Func SampleSeqTick0({tZero});
```

`tZero` If present, the new zero time to use for the `TICKS` instruction. It is not illegal to set a negative time, which may be useful if you want to take advantage of the overflow feature of the `TICKS` instruction.

Returns The zero tick time in seconds before any change made by this command was made.

Although you could collect this values using `SampleSeqVar()` (the values are stored as Spike2 clock ticks in variables `v255` and `v256`), this command greatly simplifies the process and takes care of converting the result into a time in seconds.

**See also:**

`TICKS`, `TICK0`, `SampleSeqVar()`, Sequencer variables

## SampleSequencer()

This function sets the sequencer file to attach to the Sampling configuration and sets and gets the sequencer mode (`None=0`, `Text=1` or `Graphical=2`) and can save a graphical sequence as a text sequence. Use `SampleSequencer$()` to get the name of the current sequencer file in the sampling configuration. You can also use this command to change the current sequence while sampling. There are two command variants:

```
Func SampleSequencer(name$);
Func SampleSequencer({mode%{, name$}});
```

`name$` A sequencer file name. In the first variant this sets the sequencer file name in the sampling configuration and sets `Text` mode; pass an empty string to set no sequencer file. In the second variant, `name$` is ignored unless `mode%` is 1, 3 or 4; omitting `name$` is equivalent to passing an empty string.

`mode%` If omitted, the value -1 is used. This determines what action the command take:

- 1 No effect, return the current mode.
- 0 Set sequencer setting to **None** in the sampling configuration
- 1 Set a text sequence in the sampling configuration using the name set by `name$`. If `name$` is empty, this is the same as mode 0.
- 2 Set graphical sequence mode in the sampling configuration (this is not really supported by the script as you cannot program the graphical sequencer except by loading a sampling configuration with a graphical sequence set).
- 3 Write the current graphical sequence in the sampling configuration as a text sequence to the file name in `name$`. The sequence is constructed for the 1401 type set in the resolution tab or by the `SampleOptimise()` command
- 4 Used during sampling to replace the current output sequence with the sequence held in the file named by `name$`.

Returns The first variant returns the sequencer mode at the time of the call or a negative error code. The second variant returns values depending on `mode%`:

- 1 The sequencer mode at the time of the call.
- 0 The sequencer mode at the time of the call.
- 1 The sequencer mode at the time of the call or a negative error code if the file does not exist.

- 2 The sequencer mode at the time of the call.
- 3 1=no graphical sequence, 0=success, -1=could not create file, -2=error writing the file, -3=too many varying pulses (the limit is 10).
- 4 0=success, 1=not sampling with a sequence, 2=not sampling, 3=internal error, 4=too many instructions, 5=table too big. A negative return value means that the file could not be found or did not compile to a legal sequence.

**See also:**

`SampleKey()`, `SampleOptimise()`, `SampleSequencer$()`, `SampleSeqVar()`

## SampleSequencer\$()

This function returns the name of the sequencer file that is currently attached to the sampling configuration. Use `SampleSequencer()` to set the file.

```
Func SampleSequencer$();
```

Returns It returns the current sequencer file name, or an empty string if there is no file. The returned name includes the full path.

**See also:**

`SampleKey()`, `SampleSequencer()`, `SampleSeqVar()`

## SampleSeqVar()

This is used during sampling with an output sequence, to get or set the value of an output sequencer variable. Values set before the sampling window exists are ignored. Values set before `SampleStart()` set the initial variable values.

```
Func SampleSeqVar(sVar%{, new%});
```

`sVar%` The sequencer variable to set or read, in the range 1 to 256.

`new%` The new value for the output sequencer variable. If present, the value of the variable is updated. Omit to return the variable value. A common error when setting variables for the DAC instruction is to set a value 65536 times too small.

Returns If you are setting a value, or this is used at an inappropriate time, the function returns 0. If you are reading a value, the function returns the value.

**See also:**

Output sequencer DAC instruction, `SampleSeqStep()`, `SampleSequencer$()`, `SampleStart()`

## SampleStart()

This function can be used after `FileNew()` has created a new time view based on the current Sampling configuration. It starts sampling immediately, or on a trigger.

```
Func SampleStart({trig%});
```

`trig%` 0 = start sampling immediately (default), 1 = wait for a Trigger input signal, -1 = use the sample configuration trigger setting.

Returns 0 if all went well or a negative error code.

**See also:**

`SampleAbort()`, `SampleRepeats()`, `SampleReset()`, `SampleStop()`, `SampleStartTrigger()`

## SampleStartTrigger()

This gets and optionally sets the trigger option used when sampling starts. It is equivalent to *Starting and stopping: Triggering* in the Sampling configuration Automation tab.

```
Func SampleStartTrigger({trig%});
```

`trig%` Omit this argument for no change. The following values can be used:

- 1 Use whatever state the user previously set in the sampling control panel
- 0 Sampling is not triggered
- 1 Sampling starts on a trigger; repeated files are not triggered
- 2 Sampling starts on a trigger; repeated files are triggered

Returns The trigger state (as for the `trig%` argument) at the time of the call.

**See also:**

`SampleRepeats()`, `SampleReset()`, `SampleStop()`, `SampleStart()`, `SampleStatus()`

## SampleStatus()

This function enquires about the state of any sampling.

```
Func SampleStatus();
```

Returns A code indicating the sampling state or -1 if there is no sampling:

- 0 A time view is ready to sample, but it has not been told to start yet
- 1 Sampling is waiting for an Event 3 trigger
- 2 Sampling is now in progress
- 3 Sampling is stopping (the code changes to -1 when it has stopped)

**See also:**

`SampleReset()`, `SampleStart()`, `SampleStop()`, `SampleText()`, `SampleWrite()`

## SampleStop()

This function stops sampling in progress, and is equivalent to the Stop button of the floating command window. The function does not return until sampling has stopped. If used in a file sequence, the sequence is cancelled and the current file is saved.

```
Func SampleStop();
```

Returns 0 if sampling stopped correctly or a negative error code.

**See also:**

`SampleAbort()`, `SampleReset()`, `SampleStart()`, `SampleStatus()`, `SampleWrite()`

## SampleTalker()

This function adds a Talker channel to the Sampling Configuration. If the channel is already in use, it is replaced. See the Sampling Configuration dialog for Talker details.

```
Func SampleTalker(chan%, tkr$, port%);
```

`chan%` The channel number to use for the new channel.

`tkr$` The name of the Talker (this is not case sensitive).

`port%` The data channel (defined by the Talker) to add. The first channel is 0.

Returns 0 if all went well, or a negative error code.



## SampleText()

This function adds text to a text marker channel during sampling. Text marker channels are created with the `SampleTextMark()` command.

```
Func SampleText(text$ {,time {,code%[]});
```

`text$` The text string to attach to the text marker. If the string is longer than the maximum length set for the channel, extra characters are ignored.

`time` The time for the text marker. If this argument is omitted, negative, less than the time of the last text marker, greater than the current sampling time or this is the triggered sampling mode trigger channel, then the current sampling time is used. If the time is the same as the last time written to the text marker channel, the time is incremented by one clock tick.

`code%` The first four elements of this array set the marker codes stored with the text string. If this argument is omitted the codes are set to 0. Codes are limited to the range 0-255. Only the lower 8 bits of codes outside this range are stored.

Returns 0 if all was OK, or a negative error code.

### Example

The following example is a function you could call during sampling to add a text marker with a code:

```
Func AddTextMark(text$, code%)
var cd%[4];
cd%[0] := code%; 'Set the code
return SampleText(text$, -1, cd%[]); '-1 means the current time
end;
```

### See also:

`SampleAbort()`, `SampleKey()`, `SampleReset()`, `SampleStatus()`, `SampleTextMark()`

## SampleTextMark()

This function sets channel 30 in the sampling configuration as a text marker or reads back the `TextMark` channel settings. Each event on a text marker channel holds a time, marker codes and a text string. You can add text markers to this channel using the `SampleText()` command and from a serial line. The first version of the command sets information, the second returns serial line information from the sampling configuration. The title and comment of the channel can be set with `SampleTitle$()` and `SampleComment$()`.

```
Func SampleTextMark(max%{, port%{, term${, baud%{, bits%{, par%{, stop%{, hsk%{, rate}}}}}}});
Func SampleTextMark({&term${, &baud%{ ,&bits%{, &par%{, &stop%{, &hsk %}}}}});
```

`max%` The maximum number of characters that can be attached to each text marker in the range 1 to 200. If 0 is set, the channel is deleted from the list of channels in the sampling configuration. This value is returned by the function when reading back the channel settings.

`port%` Serial port, in the range 1 to 19, to use to read on-line `TextMark` data. If omitted, no serial data is read. Use `SampleChanInfo()` to read back the port. A read back port value of 0 means no serial line input.

`term$` Optional terminating character for serial line read. If this is omitted when setting up the serial line, "\r" is used (carriage return, character code 13 or `Str$(13)`). Characters with codes less than 32 are ignored on input unless they are the terminator character.

`baud%` The serial line Baud rate (number of bits per second). The maximum character transfer rate is about one-tenth this figure. All standard Baud rates from 50 to 115200 are supported. If you do not set a Baud rate, 9600 is used.

`bits%` The number of data bits to encode a character (7 or 8). If not set, 8 data bits are used.

- `par%` Set this to 0 for no parity check, 1 for odd parity or 2 for even parity. If you do not set parity, 0 is used for no parity.
- `stop%` The number of stop bits as 1 or 2. If not set, 1 stop bit is used.
- `hsk%` The handshake mode, sometimes called flow control. 0= no handshake, 1=hardware handshake, 2=XON/XOFF protocol. If not set, 0 is used (no handshake).
- `rate` Expected sustained average data rate in text markers per second. If omitted, the rate is set to 1.

Returns 0 or a negative error code when setting the channel. When reading the settings, the return value is the maximum number of characters to store in the text marker, or 0 if the channel is not enabled.

The rate is somewhat awkwardly placed. If you want to set the rate to something other than 1 and are not using the serial input you must still set reasonable serial parameters. For example, to set 100 characters at 200 Hz without using the serial input you could use:

```
SampleTextMark(100, 0, "\r", 9600, 8, 0, 1, 0, 200.0);
```

**See also:**

`SampleChanInfo()`, `SampleComment$()`, `SampleClear()`, `SampleTitle$()`, `SampleText()`

## SampleTimePerAdc()

This sets and gets the number of time units set by `SampleUsPerTime()` for each ADC conversion. The product of the time per ADC and the microseconds per unit time must not exceed the capabilities of the 1401 you are going to use, see the **Sampling Configuration** dialog for details. Lower values will cause sampling to fail. If the optimise method is set by `SampleOptimise()` is not 0, this call will have no effect as time per ADC will be recalculated.

```
Func SampleTimePerAdc({new%});
```

- `new%` The number of clock ticks per conversion in the range 1 to 32767. If this is omitted the value is not changed. Illegal values stop the script with a fatal error.

Returns The value of the time per ADC convert at the time of the call.

**See also:**

`SampleUsPerTime()`, `SampleWaveform()`, `SampleWaveMark()`

## SampleTitle\$()

This gets and sets the title attached to a channel in the **Sampling Configuration** dialog.

```
Func SampleTitle$(chan% {,new$});
```

- `chan%` The channel number.

- `new$` If present, the new title. If the title is too long, it is truncated.

Returns The title at the time of the call, or an empty string for illegal channel numbers.

**See also:**

`SampleCalibrate()`, `SampleComment$()`, `SampleClear()`

## SampleTrigger()

This sets the four triggers that can be set to determine which data for a channel is written to disk. Each trigger is associated with a list of channels. The first trigger is also set by `SampleMode()` for backwards compatibility. You cannot change the triggering after sampling has started. Channels not associated with a trigger are written continuously.

```
Func SampleTrigger(trig%, scr%, code%, rels, relE{, cSpc});
Func SampleTrigger(trig%, get%);
```

- trig%** A trigger number in the range 1 to 4. Setting a trigger sets triggered sampling mode. This is cancelled by `SampleClear()` or by calling `SampleMode()`.
- scr%** A source channel for triggers. The channel is not checked here, so you can put in anything that could be a channel number. However, it is checked when you start sampling and if this is not an event-based channel the trigger is ignored.
- code%** If the `scr%` channel is Marker-based, you can choose to only trigger when the first marker code matches this value (in the range 0 to 255). Set -1 to accept all markers. If the source is not a marker, `code%` is ignored.
- relS** The start of the triggered area relative to the trigger time, in seconds. This can be negative or positive, but must be less than `relE`. There are limits on how far into the past a trigger can go; there is a warning message when sampling starts if the triggering requests are impossible.
- relE** The end of the triggered area relative to the trigger time. This must be greater than `relS`. Spike2 saves data in buffers so you may get more data saved to disk than you requested.
- cSpC** A channel specifier for the channels to trigger. If you omit this argument all channels are selected for triggering. You cannot use the standard channel specification values -2 or -3.
- get%** You can use the second command variant to return trigger values. Get values:
- |            |      |      |      |                                |
|------------|------|------|------|--------------------------------|
| -1         | -2   | -3   | -4   | -5                             |
| scr% code% | relS | relE | cSpC | 1=all, 1-n for one, 0=multiple |
- If more than one channel is selected, the channel specification returns 0.

Returns The first variant returns 0 or an error code. The second variant returns the requested value.

**See also:**

Channel specifiers, `SampleMode()`

## SampleUsPerTime()

This gets and optionally sets the basic time unit used for sampling. If the optimise method is set to full by `SampleOptimise(2, ...)`, changes to the basic time units have no effect as Spike2 chooses the best value to optimise the waveform sample rates.

**Func SampleUsPerTime({new});**

**new** If present, this sets the basic time unit in the range 1-1000 microseconds. Out of range values cause a fatal script error. If you sample with a Power1401 or Micro1401 mk II or -3, the value is rounded to the nearest 0.1 microseconds.

Returns The current value of microseconds per time unit.

**See also:**

`SampleOptimise()`, `SampleTimePerAdc()`

## SampleWaveform()

This function adds a waveform channel to the list of channels required. If the channel is already in use, it is replaced. The units, scale and offset fields are set to the standard Spike2 defaults (input in Volts); use `SampleCalibrate()` to change them. The title and comment of the channel can be set with `SampleTitle$()` and `SampleComment$()`.

**Func SampleWaveform(chan%, port%, ideal);**

**chan%** The channel number to use for the new channel in the range 1 to 100.

**port%** An unused (by a waveform channel) 1401 waveform port in the range 0-63.

**ideal** The ideal sampling rate that you would like for the port in Hz. Remember that you may not get this rate. Spike2 will set the nearest rate it can.

Returns 0 if all went well, or a negative error code.

**See also:**

`SampleCalibrate()`, `SampleChanInfo()`, `SampleComment$()`, `SampleClear()`,  
`SampleTimePerAdc()`, `SampleTitle$()`, `SampleUsPerTime()`, `SampleWaveMark()`

## SampleWaveMark()

This command adds a WaveMark channel to the sampling configuration and returns information about a WaveMark channel. See the sampling configuration dialog for details. The units, scale and offset fields are set to the Spike2 defaults (input in Volts); use `SampleCalibrate()` to set this. The title and comment of the channel can be set with `SampleTitle$()` and `SampleComment$()`. See the Sampling Configuration dialog for sample rate details. The first command version sets the channel, the second returns information:

```
Func SampleWaveMark(chan%, port%|port%[], rate, size%, pre%{, ideal{, nTr%});
Func SampleWaveMark(chan%{, &pre%{, &nTr%});
```

`chan%` The channel number of the WaveMark channel in the range 1 to 100.

`port%` The 1401 waveform port in the range 0-127 or an array holding the ports to use. If an array is supplied, the ports are copied from the first `nTr%` elements. If the array is too short, ports for array index `i%` are set to `port%[0] + i%`.

`rate` The estimated maximum sustained spike rate, used to allocate buffer space.

`size%` The number of waveform samples to save for each WaveMark. This must be an even number in the range 6 to 126.

`pre%` The number of points before the peak/trough of each spike, range 0 to `size%-1`;

`ideal` If present, this sets the ideal sample rate, in samples per second, for all WaveMark channels. This must be in the range 0 to 500000.0 Hz. Rates of 1 Hz or less are ignored and do not change the WaveMark rate. We suggest that you use the value 0 to leave the rate unchanged.

`nTr%` If present, this sets the number of traces to sample as 1, 2 or 4. The default is 1.

Returns The setup call returns 0 for success or a negative error code. The get information call returns `size%` (points per trace) or 0 if this is not a WaveMark channel.

You can use `SampleChanInfo()` to read back the `port%`, `rate` and `ideal` fields.

**See also:**

`SampleCalibrate()`, `SampleChanInfo()`, `SampleComment$()`, `SampleClear()`,  
`SampleTimePerAdc()`, `SampleTitle$()`, `SampleUsPerTime()`, `SampleWaveform()`

## SampleWrite()

This command controls saving data to the file during sampling. It can be used when the output data file exists, even before sampling starts. The command controls two things:

### Overall channel save data policy

Calls to this command with 1 or two arguments set the overall save data policy for one or all channels. If you enable or disable all channels, this will set or clear the Write to Disk check box in the Sample control bar. If you disable a channel, no data will be saved until you enable it again or mark a range to be saved (see below). You can set the channel save policy as soon as you create a file ready to sample and change it at any time. Changes are applied from the current sample time and overwrite any range-specific save commands.

### Mark a specific time range to be saved

Calls to this command with 3 or more arguments (a new feature in Spike2 version 8.00) mark a range of data on one or more channels to be saved to disk. You can mark time ranges in the future or in the recent past. Spike2 buffers several MB of data, so you can make retrospective decisions about keeping data. In continuous

sample mode this is only effective if writing to disk is set to paused. Time ranges can only be set after sampling has started as the initial save state for each channel is set when sampling starts and this wipes all range-based save commands. Ranges set by this command and by timed or triggered sampling are logically ORed together, that is, overlapped ranges are merged. Time ranges are used exactly with 64-bit files; with 32-bit files they mark 32kB sized blocks for saving.

```
Func SampleWrite(write% {,chan%|chan%[]{, from{, upto}}});
```

`write%` This determines the action taken by the command:

- 1 Report on the write state of the channel (or channels) given by the next argument.
- 0 Disable writing to disk (pause) the channels given by the next argument
- 1 Enable writing to disk (un-pause) the channels given by the next argument
- 2 If writing is enabled, mark a range defined by `from` and `upto` for saving to disk (for Triggered or Timed modes). This is pointless in Continuous mode as data is already being saved.
- 3 Mark the range defined by `from` and `upto` for saving to disk even if writing is disabled (for use in Continuous mode with save disabled)

`chan%` This sets the channels to operate on. If this is an integer, it is the channel number (0 or less for all channels). If it is an array, index 0 holds the number of channels following; the remaining elements hold a list of channel numbers. If omitted, all channels are used.

`from` Only used with `write%` as 2 and 3 to set the start time of a range to be saved to disk. Time ranges for saving can be set in the past (as far back as buffering allows) and in the future. Negative times are treated as zero.

`upto` Only used with `write%` as 2 and 3 to set the time up to which data is saved. If omitted or negative, saving continues to the end of the file.

Returns The overall state of writing for the channel or channels:

- 0 Disabled
- 1 Enabled
- 2 Some channels in list are enabled

## Interaction with data flushing

If you use the `Modified(0,0)` command or the Sampling configuration Automation tab Flush to disk option, you then cannot mark time ranges for saving that fell before the last saved data. Put another way, these flushing options will write all buffered data marked for writing. For example, if writing is off for a channel apart from a section from 100 to 110 seconds, and all this data from 0 up to the current time (say 200 seconds) is in the buffer and a flush option is used, the section from 100 to 110 seconds will be written to the disk. Although you can still use the data from 0 to 100 seconds within Spike2 (it is still in the buffer), you cannot now mark it for saving. Data from 110 seconds onwards can still be marked for saving.

## Level event channels

At the current time, level event data is always saved to ensure that we keep track of high and low levels correctly. This may be addressed in a future release where we make sure that we delete an even number of edges.

### See also:

`SampleAbort()`, `SampleMode()`, `SampleReset()`, `SampleStart()`, `SampleStop()`, `Sampling mode`

## ScriptBar()

This controls the Script toolbar. Call the command with no arguments to return the number of toolbar buttons. The first button is numbered 0. Script bar strings are saved in the system registry; see the `Profile()` command for more information.

```
Func ScriptBar({nBut%{, &get$});
Func ScriptBar(set$);
```

`nBut%` Set -1 and omit `get$` to clear all buttons and return 0. Otherwise it is a button number and returns -1 if the button does not exist, 0 if it is the last button, and 1 if higher-numbered buttons exist. `get$` returns the information as for `set$`.

`set$` This holds up to 8 characters of button label, a vertical bar, the path to the script file including `.s2s`, a

vertical bar and a pop-up comment. The function returns the new number of buttons or -1 if all buttons are already used.

Returns See the descriptions above. Negative return values indicate an error.

For example, the following code clears the script bar and sets a button:

```
ScriptBar(-1); 'clear all buttons
ScriptBar("ToolMake|C:\\Scripts\\ToolMake.s2s|Build a toolbar");
```

**See also:**

App(), Script toolbar

## ScriptRun()

This sets the name of a script to run when the current script terminates. You can pass information to the new script using disk files or by using the Profile() command. You can call this function as often as you like; only the last use has any effect.

```
Proc ScriptRun(name${, flags%});
```

**name\$** The script file to run. You can supply a path relative to the current folder or a full path to the script file. If you supply a relative path, it must still be valid at the end of the current script. Set name\$ to " " to cancel running a script.

**flags%** Optional flags, taken as 0 if omitted. Sum of: 1 = run even if the current script ends in an error, 2 = keep loaded script in memory

If the file you name does not exist when Spike2 tries to run it, nothing happens. If the nominated script is not already loaded, Spike2 will load it, run it and unload it unless the keep loaded script in memory flag is set. If a loaded script calls Yield() or calls any function that allows the system to idle (ToolBar(), DlgShow()...), the script can be unloaded while it is still running. This is usually harmless unless the loaded script attempts to use App(3), which will return 0 if the script is no longer in memory.

**See also:**

App(), Profile()

## Seconds()

This returns the time in seconds. This is used for relative time measurements. You can also set the timer. If you want the time into sampling, use the Maxtime() function. To find out how much time built-in script functions are using, see DebugList().

```
Func Seconds({set{, hiRes}});
```

**set** If present, this sets the time in seconds. The time is 0 when Spike2 starts.

**hiRes** If present, this selects between normal or high-resolution timing. A zero value sets the standard resolution of nominally 1 millisecond (but may be worse on some systems). Set 1 for the highest resolution available. Note that the default is 1 millisecond resolution, set when Spike2 starts. Changes to the resolution are persistent between scripts.

Returns If hiRes is not present, the function returns the time in seconds. This is the value before any new time is set. If hiRes is present, the return value is the time resolution, in seconds. This is 1 millisecond for the standard resolution and can be (much) less than 1 microsecond for the high resolution timer.

### High resolution timer caveats

The function we use for the high-resolution timer is supposed to use a fixed frequency source. However, some machines (incorrectly) use a time based on CPU cycles; this is a problem for many laptops which change the CPU speed to save battery power. There are also reports that some multi-core machines report different times for each core. If you have a problem, the following web links may be useful: South bridge chipset problem, Athlon X2 problem. Note that computers from 2007 onwards are less likely to have problems.

**See also:**

Date\$(), DebugList(), MaxTime(), Time\$(), TimeDate()

## Selection\$()

This function returns the text in the current view that is currently selected.

```
Func Selection$();
```

Returns The current text selection. If there is no text selected, or if the view is inappropriate for this action, an empty string is returned.

**See also:**

EditCopy(), EditCut(), EditPaste(), MoveBy(), MoveTo()

## SerialClose()

This function closes a serial port opened by `SerialOpen()`. Closing a port releases memory and system resources; any characters held in the `SerialWrite()` output buffer waiting to be transmitted will be lost. Ports are automatically closed when a script ends, however it is good practice to close a port when your script has finished with it.

```
Func SerialClose(port%);
```

port% The serial port to close as defined for `SerialOpen()`.

Returns 0 or a negative error code.

You can find the size of the `SerialWrite()` output buffer by using `SerialWrite(port%)` immediately after `SerialOpen()`, then monitor buffered characters before using `SerialClose()` if loss of buffered characters is an issue in your application.

**See also:**

SerialOpen(), SerialWrite(), SerialRead(), SerialCount()

## SerialCount()

This counts the characters or items buffered in a serial port opened by `SerialOpen()`. Use this to detect input so your script can do other tasks while waiting for serial data. There is an internal buffer of 1024 characters per port that is filled when you use `SerialCount()`. The size of this buffer limits the number of characters that this function can tell you about. To avoid character loss when you are not using a serial line handshake, do not buffer up more than a few hundred characters with `SerialCount()`.

```
Func SerialCount(port% {,term$});
```

port% The serial port to use as defined for `SerialOpen()`.

term\$ An optional string holding the character(s) that terminate an input item.

Returns If `term$` is absent or empty, this returns the number of characters that could be read. If `term$` is set, this returns the number of complete items that end with `term$` that could be read.

**See also:**

SerialOpen(), SerialWrite(), SerialRead(), SerialClose()

## SerialOpen()

This function opens a serial port and configures it for use by the other serial line functions. It is not an error to call `SerialOpen` more than once on the same port. The serial routines use the host operating system serial line support. Consult your system documentation for information on serial line connections and Baud rate limits.

```
Func SerialOpen(port%{, baud%{, bits%{, par%{, stop%{, hsk%}}}}});
```

- port%** The serial port to use, in the range 1 to 256. The number of ports depends on the computer. Modern machines usually only implement serial interfaces through USB plug-in devices and the port numbers are device dependent. The maximum port number was 9 before Spike2 version 7.09.
- baud%** This sets the serial line Baud rate (number of bits per second). The maximum character transfer rate is of order one-tenth this figure. All standard rates from 50 to 115200 Baud are supported. If you omit **baud%**, 9600 is used.
- bits%** The number of data bits to encode a character. Windows supports 4 to 8 bits, the Macintosh supports 7 or 8. If **bits%** is omitted, 8 is set. Standard values are 7 or 8 data bits. If you set 7 data bits, character codes from 0 to 127 can be read. If you set 8 data bits, codes from 0 to 255 are possible.
- par%** Set this to 0 for no parity check, 1 for odd parity or 2 for even parity. If you do not specify this argument, no parity is set.
- stop%** This sets the number of stop bits as 1 or 2. If omitted, 1 stop bit is set. If you specify 5 data bits, a request for 2 stop bits results in 1.5 stop bits being used.
- hsk%** This sets the handshake mode, sometimes called "flow control". 0 sets no handshake, 1 sets a hardware handshake, 2 sets XON/XOFF protocol.

Returns 0 or a negative error code.

#### See also:

`SerialWrite()`, `SerialRead()`, `SerialCount()`, `SerialClose()`

## SerialRead()

This function reads characters, a string, an array of strings, or binary data from a nominated serial port that was previously opened with `SerialOpen()`. Binary data can include character code 0; string data never includes character 0.

```
Func SerialRead(port%, &in$|in$[]|&in%|in%[]{,term${, max%}});
```

- port%** The serial port to read from as defined for `SerialOpen()`.
- in\$** A single string or an array of strings to fill with characters. To use an array of strings you must set a terminator or all input goes to the first string in the array.
- in%** A single integer (**term\$** and **max%** are ignored) or an array of integers (**term\$** and **max%** can be used) to read binary data. Each integer can hold one character, coded as 0 up to 255. The function returns the number of characters returned.
- term\$** If this is an empty string or omitted, all characters read are input to the string, integer array or to the first string in the string array and the number of characters read can be limited by **max%**. The function returns the number of characters read.
- If **term\$** is not empty, the contents are used to separate data items in the input stream. Only complete items are returned and the terminator is not included. For example, set the terminator to "\n" if lines end in line feed, or to "\r\n" if input lines end with carriage return then line feed. If **in\$** is a string, one item at most is returned. If **in\$[]** is an array, one item is returned per array element. The function returns the number of items read unless **in** is an integer (**in%** or **in%[]**) when it returns the number of characters returned.
- max%** If present, it sets the maximum number of characters to read into each string or into the integer array. If a terminator is set, but not found after this many characters, the function breaks the input at this point as if a terminator had been found. There is a maximum limit set by the integer array size, the size of the buffers used by Spike2 to process data and by the size of the system buffers used outside Spike2. This is typically 1024 characters.

Returns The function returns the number of characters or items read or a negative error code. If there is nothing to read, it waits 1 second for characters to arrive before timing out and returning 0. Use `SerialCount()` to test for items to read to avoid a time out.



**See also:**

`SerialOpen()`, `SerialWrite()`, `SerialCount()`, `SerialClose()`

## SerialWrite()

This writes one or more strings or binary data to a serial port opened by `SerialOpen()`. Use the command with a single argument to find out how much space is available in the `SerialWrite()` output buffer (typically 1024 characters).

```
Func SerialWrite(port%{, out$|out$[]|out%|out%[][, term$}});
```

`port%` The serial port to write to as defined for `SerialOpen()`.

`out$` A single string or an array of strings to write to the output. The return value is the number of strings written.

`out%` A single integer or an integer array to write as binary. One value is written per integer. The output written depends on the number of data bits set for the port; 7-bit data writes as `out% band 127`, 8-bit data writes as `out% band 255`. The return value is 1 if the transfer succeeded.

`term$` If present, it is written to the output port after the contents of `out%`, `out%[]` or `out$` or after each string in `out$[]`.

**Returns** For success, the return value is as documented for the `out` argument. If there is no room in the output buffer for the data the return value is -1 except when `out$[]` is used when the return values is the count of strings actually sent. When called with a single argument, the command returns the number of free characters in the output buffer.

If you use the `SerialClose()` command before the system has had time to write buffered characters to the serial port, the buffered characters will be lost.

**See also:**

`SerialOpen()`, `SerialRead()`, `SerialCount()`, `SerialClose()`

## Set...()

The `Set...()` family of commands create result views attached to the current time view or to the time view associated with the current result view. Apart from `SetResult()`, which creates a result view that is not dependent on a time view, these routines match the Analysis menu New Result view commands. They do not update the display (use `Draw()` or `DrawAll()`), nor do they perform any analysis (see the `Process()` command).

The commands return a view handle, or a negative error code. Errors include: Bad channel number, illegal number of bins, out of memory, illegal bin size. The new view is made the current view. However, it is created invisibly and must be made visible with `WindowVisible(1)` before it will appear when drawn.

Most commands accept a channel specifier, written as `cSpC`. It can be an integer channel number, or -1 for all, -2 for visible or -3 for selected channels. It can also be a channel specification string, such as "1..4,6,10" or an integer array where the first element is the channel count, the remaining elements are the channel numbers. Whatever channels are specified, only channels of the correct type for the command are used. If a channel is included twice, the first occurrence is used. It is an error if the resulting list is empty. Result view channels appear in the same order as in the specification. The first channel in `cSpC` generates result channel 1, the second generates channel 2, and so on.

**Choose a command for more information:**

`SetAverage()`, `SetEvtCrl()`, `SetEvtCrlShift()`, `SetINTH()`, `SetPhase()`, `SetPower()`, `SetPSTH()`, `SetResult()`, `SetWaveCrl()`, `SetWaveCrlDC()`

## SetAverage()

This command creates a result view to hold the sum or average of sweeps of waveform data and makes it the current view. The `Process()` command does the analysis. `Sweeps()` reports the number of sweeps accumulated. Omitted optional arguments have the value 0. The function is:

```
Func SetAverage(cSpc, bins% {,offset {,trig% {,flags%{,align%}}});
```

**cSpc** A channel specifier of waveform or WaveMark channels to average in the current time view, or in the time view associated with the current result view. Invalid channels are ignored. The channel list must hold at least one valid channel. All the channels must have the same sampling rate; any channels that do not match the sample rate of the first channel in the list are ignored.

**bins%** The number of bins required. There must be at least one bin. The maximum is limited by available memory. The bin width is the waveform sampling interval.

**offset** This sets the pre-trigger time to show in the result. If omitted, 0 is used.

**trig%** The channel number to use as a trigger for each sweep. If this is omitted, or set to 0, then each call to `Process()` takes the start time as the trigger time.

**flags%** This is the sum of flag values: 1=display the mean data and not the sum, 4=enable error bars, 32=count items per bin. If `flags%` is omitted, 0 is used.

**align%** How to align data to the trigger. 0=Next point, 1=nearest point, 2=linear interpolation, 3=cubic spline interpolation. If omitted, 0 is used.

**Returns** The function returns a handle for the new view, or a negative error code.

### See also:

Channel specifiers, `BinError()`, `ChanData()`, `DrawMode()`, `Process()`, `Sweeps()`

## SetEvtCrl()

This creates a result view of event correlation histograms and optional raster displays and makes it the current view. The `Process()` command does the analysis. `Sweeps()` reports the number of triggers processed. You can take measurements from an auxiliary channel from each sweep and use this value to sort rasters, display a symbol and/or discard sweeps. See the Event correlation in the Analysis menu for details. See `RasterSort()` and `RasterSymbol()` for a description of sort values and symbol times.

```
Func SetEvtCrl(cSpc, bins%, binsz{, offset{, trig%{, flags%{, aCh%{, Mn, Mx}}}});
```

**cSpc** A channel specifier of event or marker channels in the current time view, or in the time view associated with the current result view. Invalid channels are ignored. The resulting channel list must hold at least one valid channel.

**bins%** The number of bins in the histogram. There must be at least 1 bin.

**binsz** The width of each bin in seconds. This is converted into underlying time units.

**offset** This sets the pre-trigger time to show in the histogram.

**trig%** The channel number to use as a trigger for each sweep. If this is omitted, or set to 0, then each call to `Process()` takes the start time as the trigger time.

**flags%** This is the sum of: 1 to scale the result as spikes per second, 2 to enable raster displays, 8 for backwards `aCh%` event search, 16 to exclude rather than include values that lie in the `Mn` to `Mx` range. If omitted, 0 is used.

**aCh%** Auxiliary measurement channel. The measured value sets raster sort value 1. For waveform or RealWave channels, this is the waveform value at the trigger time. For all other channel types, it is the latency of the first event on the channel before/after the trigger and it also sets the time of symbol 1. If you omit `aCh%`, there is no auxiliary channel.

**Mn, Mx** If present, these arguments set the range of measured values from `aCh%` that control if a sweep is

included or excluded.

Returns The function returns a handle for the new view, or a negative error code.

For an auto-correlation, set `trig%` the same as `chan%`. In this case, we do not count the result of correlating each event with itself. If you must count self-correlations, add `Sweeps()` to the bin holding the time shift of zero.

#### See also:

Event time cross correlogram, Channel specifiers, `SetEvtCrlShift()`, `DrawMode()`, `Process()`, `RasterSet()`, `RasterSort()`, `RasterSymbol()`, `Sweeps()`

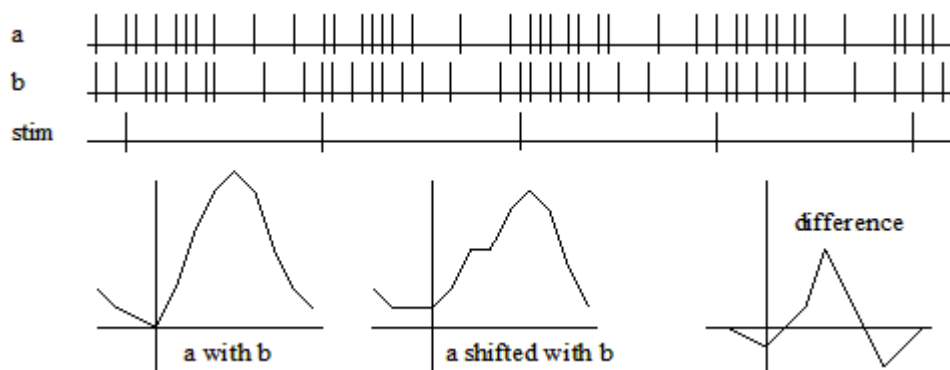
## SetEvtCrlShift()

This can be used when the current result view was created by `SetEvtCrl()`. It sets a time shift for the next `Process()` command to produce shuffled correlations by shifting one channel with respect to the other without changing the result view time axis.

**Proc SetEvtCrlShift(shift);**

`shift` This value can be positive or negative. It causes the data on the `trig%` channel to be correlated with data from channel `chan%` at a time `shift` later. The shift is set back to zero after the `Process()` command on this result view.

Consider two channels of events (to be correlated) and a channel marking stimuli:



If the stimulus channel holds events at some constant interval `int`, by shifting all events on one of the two channels by `int`, the only correlation between the two channels left is the correlation due to the stimulus. Thus by forming the difference between the unshifted correlation and the shifted correlation, you can obtain the correlation between `a` and `b` with the effect of the stimulus removed.

#### See also:

`SetEvtCrl()`, `Process()`

## SetINTH()

This function creates a result window to hold an interval histogram and makes it the current view. The `Process()` command does the analysis. Each interval processed increments the value returned by `Sweeps()`, even intervals that are too short or too long to contribute to the histogram.

**Func SetINTH(cSpc, bins%, binsz{, minInt});**

`cSpc` A channel specifier of event or marker channels in the current time view, or in the time view associated with the current result view. Invalid channels are ignored. The resulting channel list must hold at least one valid channel.

`bins%` The number of bins required. There must be at least one bin. The maximum is limited by available memory.

`binsz` The width of each bin in seconds. This is converted into underlying time units.

`minInt` This sets the start of the first analysis bin, in seconds. Shorter intervals are counted for `Sweeps()`, but not included in the histogram. If omitted, 0 is used.

Returns The function returns a handle for the new view, or a negative error code.

**See also:**

Channel specifiers, `Process()`, `Sweeps()`

## SetPhase()

This function creates a result view to hold phase histograms with optional raster displays and makes it the current view. The `Process()` command does the analysis. `Sweeps()` reports the number of cycles analysed. You can measure a value for each sweep from an auxiliary channel and use this value to sort rasters, display a symbol and discard sweeps. See the description of the Analysis menu Phase histogram for details. See `RasterSort()` and `RasterSymbol()` for a description of sort values and symbol times.

```
Func SetPhase(cSpc, bins%, minCyc{, maxCyc{, cycle%{, flags%{, aCh%{, Mn, Mx}}}});
```

`cSpc` A channel specifier of event or marker channels in the current time view, or in the time view associated with the current result view. Invalid channels are ignored. The resulting channel list must hold at least one valid channel.

`bins%` The number of bins required. There must be at least one bin. The maximum is limited by available memory. The bins are each  $360/bins\%$  wide.

`minCyc` The minimum cycle time to use in seconds. If two consecutive cycle markers are closer than this time, the cycle is ignored, and is not counted by `Sweeps()`. If this is omitted, a value of 0.0 is used.

`maxCyc` The maximum cycle time to use in seconds. If two consecutive cycle markers are further apart than this, the cycle is ignored, and is not counted by `Sweeps()`. If this is omitted, there is no limit on cycle size.

`cycle%` The channel number to use as the cycle start and end marker. If this is omitted, or set to 0, then each call to `Process()` takes the start time and end time as marking a single cycle.

`flags%` This is the sum of: 2 to enable raster displays, 8 for backwards `aCh%` event search, 16 to exclude rather than include sweeps with measured values that lie in the `Mn` to `Mx` range. If omitted, 0 is used.

`aCh%` Auxiliary measurement channel. The measured value sets raster sort value 1. For waveform or `RealWave` channels, this is the waveform value at the cycle start. For all other channel types, it is the latency of the first event on the channel before/after the cycle start and it also sets the symbol 1 time. If you omit `aCh%`, there is no auxiliary channel.

`Mn, Mx` If present, these arguments set the range of measured values from `aCh%` that control if a sweep is included or excluded.

Returns The function returns a handle for the new view, or a negative error code.

**See also:**

Event phase histogram, Channel specifiers, `DrawMode()`, `SetEvtCrl()`, `Process()`, `RasterSet()`, `RasterSort()`, `RasterSymbol()`, `Sweeps()`

## SetPower()

This function creates a result view to hold a power spectrum and makes it the current view. The `Process()` command does the analysis. The function is as follows:

```
Func SetPower(cSpc, fftsz% {,wnd%});
```

`cSpc` A channel specifier of waveform channels in the current time view, or in the time view associated with the current result view. Invalid channels are removed from the list. The resulting channel list must hold at least one valid channel. All the channels must have the same sampling rate; any channels that do not match the sample rate of the first channel in the list are ignored.

`fftsz%` The size of the transform used in the FFT. This must be a power of 2 in the range 16 to 262144. The result view has half this number of bins. The width of each bin is the sampling rate of the channel divided by `fftsz%`. Each block of `fftsz%` data points processed increments the value for `Sweeps()`.

`wnd%` The window to use. 0 = none, 1 = Hanning, 2 = Hamming. Values 3 to 9 set Kaiser windows with -30 dB to -90 dB sideband ripple in steps of 10 dB. If this is omitted a Hanning window is applied.

Returns The function returns a handle for the new view, or a negative error code.

**See also:**

Channel specifiers, `ArrFFT()`, `Process()`, `Sweeps()`

## SetPSTH()

This creates a result view to hold a peri-stimulus time histogram with an optional raster display and makes it the current view. The `Process()` command does the analysis. `Sweeps()` returns the number of triggers processed. You can measure a value for each sweep from an auxiliary channel and use this to sort rasters, display a symbol and discard sweeps. See the description of the PSTH in the Analysis menu for details. See `RasterSort()` and `RasterSymbol()` for a description of sort values and symbol times.

```
Func SetPSTH(cSpc, bins%, binsz {,offset {,trig% {,flags%{,aCh%
{,Mn,Mx}}}});
```

`cSpc` A channel specifier of event or marker channels in the current time view, or in the time view associated with the current result view. Invalid channels are ignored. The resulting channel list must hold at least one valid channel.

`bins%` The number of bins required. There must be at least one bin.

`binsz` The width of each bin in seconds. This is converted into underlying time units.

`offset` This sets the pre-trigger time, in seconds. If omitted, 0.0 is used.

`trig%` The channel number to use as a trigger for each sweep. If this is omitted, or set to 0, then each call to `Process()` takes the start time as the trigger time.

`flags%` This is the sum of: 1 to scale the result as spikes per second, 2 to enable raster displays, 8 for backwards `aCh%` event search, 16 to exclude rather than include values that lie in the `Mn` to `Mx` range. If omitted, 0 is used.

`aCh%` Auxiliary measurement channel. The measured value sets raster sort value 1. For waveform or `RealWave` channels, this is the waveform value at the trigger time. For all other channel types, the value is the latency of the first event on the channel before/after the trigger and it also sets the symbol 1 time. If you omit `aCh%`, there is no auxiliary channel.

`Mn, Mx` If present, these arguments set the range of measured values from `aCh%` that control if a sweep is included or excluded.

Returns The function returns a handle for the new view, or a negative error code.

**See also:**

Peri-Stimulus time histogram, Channel specifiers, `DrawMode()`, `SetEvtCrl()`, `Process()`, `RasterAux()`, `Sweeps()`

## SetResult()

This function creates a result view of user-defined type, attached to no time view and with no implied `Process()`. It becomes the current view. `Sweeps()` will return 0 unless you set the sweep count.

```
Func SetResult({chans%,} bins%, binsz, offset, title$, xU$ {, yU$ {,xT$
{,yT$ {,flags% {,tick}}}});
```

`chans%` This sets number of channels in the new result view. Omit for 1 channel. See the `View()` documentation for access to multiple channel result view data.

- `bins%` The number of bins in the view.
- `binsz` The width of each bin. Bins should have a positive non-zero width.
- `offset` The x axis value at the start of the first bin.
- `title$` The window title.
- `xU$` The x axis units.
- `yU$` Optional, y axis units, blank if omitted.
- `xT$` Optional, x axis title (otherwise blank).
- `yT$` Optional, y axis title (otherwise blank).
- `flags%` Add 1 for mean (not sum) mode for errors, 2 to enable raster data, 4 to enable error bars, 32 for individual counts for each bin. The default value is 0.
- `tick` This is required if you enable raster data. Result views store raster data as 32-bit integers. `tick` is the time resolution for this data. When working with a time view, set `tick` to `BinSize()`, the time resolution of the time view. The maximum time of a raster event is  $2147483647 * tick$  seconds.

Returns The function returns a handle for the new view, or a negative error code.

**See also:**

`BinError()`, `DrawMode()`, `RasterGet()`, `RasterSet()`, `Sweeps()`, `View()`

## SetWaveCrl()

This creates a result view to hold the waveform correlation between waveform channels sampled at the same rate and makes it the current view. The `Process()` command does the analysis. `Sweeps()` returns the number of data points used on the reference channel.

```
Func SetWaveCrl(cSpc, ref%, bins%{, offset});
```

- `cSpc` A channel specifier of waveform channels in the current time view, or in the time view associated with the current result view. Invalid channels are removed from the list. There must be at least one valid channel. Channels that do not match the sample rate of the first channel in the list are ignored.
- `ref%` A reference waveform channel that is correlated with all the waveform channels.
- `bins%` The number of bins in the correlation. The bin width is the channel sampling interval. This calculation is slow compared to the other `SetXXXX` commands. The time taken is proportional to the length of the area processed times `bins%`.
- `offset` The time to show before the zero time shift, in seconds. If omitted, 0 is used.

Returns The function returns a handle for the new view, or a negative error code.

**See also:**

Channel specifiers, `SetWaveCrlDC()`, `Process()`, `Sweeps()`

## SetWaveCrlDC()

To use this function the current view must be a waveform correlation result view. It sets whether the DC levels of the two signals is removed before the correlation or not.

```
Func SetWaveCrlDC({useDC%})
```

- `useDC%` If present, a zero value removes the DC, a non-zero value (default) includes it.

Returns The `useDC%` value before the call. It returns a negative error if this view is a result view but not a waveform correlation.

If you change the setting the view is drawn at the next opportunity.

**See also:**

SetWaveCrl(), Process()

## Sin()

This calculates the sine of an angle in radians, or converts an array of angles into sines.

```
Func Sin(x|x[]{|[]...});
```

**x** The angle, expressed in radians, or a real array of angles. The best accuracy of the result is obtained when the angle is in the range  $-2\pi$  to  $2\pi$ .

**Returns** When the argument is an array, the function replaces the array with the sines and returns 0 or a negative error code. When the argument is not an array the function returns the sine of the angle.

**See also:**

Abs(), ATan(), Cos(), Cosh(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sqrt(), Sinh(), Tan(), Tanh()

## Sinh()

This calculates the hyperbolic sine of a value or of an array of values.

```
Func Sinh(x|x[]{|[]...});
```

**x** The value, or an array of real values.

**Returns** When the argument is an array, the function replaces the array elements with their hyperbolic sines and returns 0. When the argument is not an array the function returns the hyperbolic sine of the argument.

**See also:**

Abs(), ATan(), Cos(), Cosh(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sqrt(), Sinh(), Tan(), Tanh()

## SMControl()

This function sets and reads the toolbar and edit field states of the current spike monitor window. See the documentation of the spike monitor window for a full description.

```
Func SMControl(item%{, new});
```

**item%** This identifies the item:

|    |                          |                                                |
|----|--------------------------|------------------------------------------------|
| 0  | Draw mode                | 0=3D, 1=2D, 2=2D separated                     |
| 1  | Fade to background       | 0=no fade, 1=fade                              |
| 2  | Colour last spike only   | 0=colour all, 1=colour last, the rest are grey |
| 3  | Use thick lines          | 0=use thin lines, 1=use thick lines (slow)     |
| 4  | Timed (smooth) update    | 0=update on new data, 1=update on timer        |
| 5  | At end mode              | 0=use cursor 0, 1=at end of file               |
| 6  | Lock y axes              | 0=track data size, 1=fix at current size       |
| 7  | Show duplicate channels  | 0=Show originals only, 1=show duplicates       |
| 8  | Maximum spikes           | In the range 1 to 40                           |
| 9  | Time range               | In the range 0.003 to MaxTime()                |
| 10 | Display rectangles       | 0=no, 1=show front and back rectangles         |
| 11 | Front rectangle x offset | In the range 0 to 0.9                          |
| 12 | Front rectangle scale    | In the range 0.1 to 1.0                        |
| 13 | Back rectangle x offset  | In the range 0 to 0.9                          |
| 14 | Back rectangle scale     | In the range 0.1 to 1.0                        |

**new** If present, this sets the state of the item. See the table for acceptable values.

Returns The item state at the time of the call or -1 if the item does not exist.

**See also:**

Spike Monitor window, `SSChan()`, `SSClassify()`, `SSOpen()`, `SSParam()`, `SSRun()`

## SMOpen()

This function gets the spike monitor window handle or opens it. The current view must be a time view or a spike monitor window. Close the spike monitor window with `FileClose()`. Use the `SMControl()` command to control the window. You can use the `Window()` and `WindowVisible()` commands to size, show and hide the window.

**Func** `SMOpen({type%{, mode%});`

`type%` This argument is assumed to be zero if omitted. Allowed values are:  
 0 Returns the view handle of the spike monitor window associated with the current view, or 0 if there is no open window.  
 1 Open the spike monitor window. Returns the handle or 0 if we failed.  
`mode%` If this is zero or omitted, the dialog is created invisibly. Set 1 to make it visible. This is ignored unless `type%` is 1.

Returns The return value is the view handle of the spike monitor window or 0.

**See also:**

Spike Monitor window, `FileClose()`, `SMControl()`, `ViewLink()`, `Window()`, `WindowVisible()`

## Sound()

This command has two variants. The first plays a .wav file or system sound through your sound card if your system has multimedia support. The second plays a tone of set pitch and duration through the motherboard speaker, which is unlikely to exist, or if it does exist is unlikely to be useful in a modern PC. If you want to play data held in a channel of a data file, see the `PlayOffline()` script command. If you want to convert text to speech, see the `Speak()` script command.

### Multimedia (sound card) output

This command variant uses a simple system feature that will play a nominated file of less than some maximum size (256 kB the last time I looked into this, but it could be less on some systems) through the multimedia sound system. As we have made no provision in Spike2 for setting this up, you must have arranged your system so that the default sound output is routed to wherever you require it.

**Func** `Sound(name${, flags%);`

`name$` The name of a .wav file or of a system sound. You can supply the full path to the file or just a file name and the system will search for the file in the current directory, the Windows directory, the Windows system directory, directories listed in the `PATH` environmental variable and the list of directories mapped in a network. If no file extension is given, .wav is assumed. The file must be short enough to fit in available physical memory.

A blank name (" )halts sound output. If `name$` is any of the following (case is important), a standard system sound plays:

|      |          |      |       |      |         |      |             |
|------|----------|------|-------|------|---------|------|-------------|
| "S*" | Asterisk | "SS" | Start | "SE" | Exit    | "SH" | Hand        |
| "SW" | Welcome  | "S?" | Query | "SD" | Default | "S!" | Exclamation |

`flags%` This optional argument controls how the data is played. It is the sum of:

|        |   |                                                                                                                                                                                     |
|--------|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x0001 | 1 | Play asynchronously (start output and return). Without this flag, the <code>Sound()</code> command waits until replay ends.                                                         |
| 0x0002 | 2 | Silence when sound not found. Normally <code>Sound()</code> plays the system default sound if the nominated sound cannot be found.                                                  |
| 0x0008 | 8 | Loop sound until stopped by another <code>Sound()</code> command; use " " for <code>name\$</code> to stop output.. You must also supply the asynchronous flag if you use loop mode. |



- 0x0010 16 Don't stop a playing sound. Normally, unless the "No wait" flag is set, each command cancels any playing sound.
- 0x2000 8192 No wait if sound is already playing. `Sound("", 0x2010)` returns 1 if a previous asynchronous sound is finished and 0 if not.

If you don't supply this argument, the flag value is set to 0x2000.

Returns The multimedia output returns non-zero if the function succeeded and zero if it failed.

### Motherboard speaker output

Most PC motherboards have a header that can be connected to a small speaker. This was originally used to signal system startup failures and is driven by a timer chip that produces a square wave output at a settable frequency. If it exists on your system it will generate a nasty noise. We suggest that you avoid using this command (and we may even remove it in future versions of Spike2).

```
Func Sound(freq%, dur{, midi%});
```

`freq%` If `midi%` is 0 or omitted this holds the sound frequency in Hz. If `midi%` is non-zero this is a MIDI value in the range 1-127. A MIDI value of 60 is middle C, 61 is C# and so on. Add or subtract 12 to change the note by one octave. This uses the system speaker (usually a small and nasty device attached to your computer motherboard), not your sound card.

`dur` The sound duration, in seconds. The script stops during output.

`midi%` If present and non-zero, `freq%` is interpreted as a MIDI value.

returns 0 or a negative error code.

### See also:

`PlayOffline()`, `Speak()`

## Speak()

If your system supports text to speech, this command allows you to convert a text string into speech. We do not provide facilities to setup voices or to route the sound output; you must do this from the Speech applet in the control panel. There are two command variants. The first outputs text as speech; the second is for control of the output and provides status information.

```
Func Speak(text${, opt%});
Func Speak({what%{, val}});
```

`text$` A string holding the text to output, for example "Sampling has started".

`opt%` This optional argument (default value 1) controls the text conversion and output method. It is the sum of the following flags:

- 1 Speak asynchronously. Without this flag the command waits until speech output is over before returning.
- 2 Cancel any pending speech output.
- 4 Speak punctuation marks in the text.
- 8 Process embedded SAPI XML. For example:  
`Speak("Emphasis on <EMPH>this</EMPH> word", 8);`
- 16 Reset to the standard voice settings before speaking.

`what%` An optional variable, taken as 0 if it is omitted:

- 0 Returns 1 if speech is playing and 0 if it is not.
- 1 Wait for up to `val` seconds (default value 3.0) for output to end. The return value is 0 if playing is finished, 1 if it continues after `val` seconds.
- 2 Returns the current speech speed in the range -10 to 10; 0 is the standard speed. If `val` is present, it sets the new speed.
- 3 Returns the current speech volume in the range 0 to 100, 100 is the standard volume. If `val` is present, it sets the new volume.

`val` An optional argument used when `what%` is greater than 0.

Returns If there is no speech support available, or a system error occurs, the command returns -1. Otherwise

the first command variant returns 0 if all is well and the second variant returns the values listed for what%.

To use TTS (text to speech), you need a suitable sound card and the Microsoft SAPI software support. Windows XP onwards has this software included with the operating system. You can get text to speech support as a download for earlier versions of Windows. In August 2006, the speech support was available as SpeechSDK51MSM.exe from the web page [www.microsoft.com/speech/download/sdk51/](http://www.microsoft.com/speech/download/sdk51/) but this location may change.

**See also:**

Sound(), Embedded SAPI XML

## Embedded SAPI XML

It is possible to embed instructions to the "voice" within the text to change how the text is rendered. Here are some simple examples of embedded XML:

```
Speak("<EMPH>This</EMPH> is an emphasis", 8);
Speak("<RATE SPEED='-5'>This is slow speech.</RATE>", 8);
Speak("<PITCH MIDDLE='5'>This is high pitched speech.</PITCH>", 8);
Speak("<VOLUME LEVEL='50'>This is quiet speech.</VOLUME>", 8);
Speak("<SPELL>Spell this out.</SPELL>", 8);
Speak("Five hundred milliseconds of silence <silence msec='500' /> just occurred.", 8);
```

There are other more complicated commands you can give. See this web site for more information on this advanced topic.

## Sqrt()

Forms the square root of a real number or an array of real numbers. Negative numbers halt the script with an error when x is not an array. With an array, negative numbers are set to 0 and an error is returned.

```
Func Sqrt(x|x[]{|[]...});
```

x        A real number or a real array to replace with an array of square roots.

Returns With an array, this returns 0 if all was well, or a negative error code. With an expression, it returns the square root of the expression.

**See also:**

Abs(), ATan(), Cos(), Exp(), Frac(), Ln(), Log(), Max(), Min(), Pow(), Rand(), Round(), Sin(), Tan(), Trunc()

## SS...()

This family of commands gives you script control over the Spike Shape dialog. You open the dialog, or get the handle to an already open dialog with the SSOpen() command. You get and set the current channel with SSChan() and step or run with the SSRun() command. Use SSParam() to access the template parameters dialog. The SSTemp...() command family gives you access to the templates. The SSButton() command lets you set and get button and check box states in the dialog that control how it operates. The HCursor() command controls the horizontal cursors. The SSCol...() commands give you script control over collision analysis. Use FileClose() to shut the dialog.

The following commands are also implemented for Spike shape views:

- ViewLink()            Find the parent time view of the spike shape dialog.
- Optimise()            Use with no arguments to optimise the display, equivalent to the Automatic Levels button.
- YRange()              Sets the displayed y axis range or set the display to full range.

|                                            |                                                                             |
|--------------------------------------------|-----------------------------------------------------------------------------|
| <code>YHigh()</code> , <code>YLow()</code> | Get the y axis limits.                                                      |
| <code>XRange()</code>                      | Sets the time view range to process, equivalent to Time Range button        |
| <code>XLow()</code> , <code>XHigh()</code> | Gets the start and end of the time range to process.                        |
| <code>Cursor(0)</code>                     | Use this in the associated Time view to find the time of the current spike. |
| <code>WindowVisible()</code>               | Show, hide and maximise the window.                                         |
| <code>Window()</code>                      | Position the dialog                                                         |
| <code>WindowGetPos()</code>                | Find the dialog position                                                    |
| <code>WindowTitle\$( )</code>              | Get and set the dialog title                                                |

## SSButton()

This function sets and reads the toolbar and check box states of the current spike shape window. See the documentation of the spike shape dialogs for a description of the items.

```
Func SSButton({item%{, new%}});
```

`item%` If omitted, all items are reset to 0. Otherwise this identifies the item:

|    |                                    |                                  |
|----|------------------------------------|----------------------------------|
| 0  | Overdraw spikes                    | 0=no overdraw, 1=overdraw        |
| 1  | Show template boundaries           | 0=hide, 1=show                   |
| 2  | Show non-matching spikes           | 0=hide, 1=show                   |
| 3  | Play sound for each spike          | 0=no sound, 1=sound              |
| 4  | Scroll time view to track cursor 0 | 0=no track, 1=track              |
| 5  | At end mode (online)               | 0=not at end, 1=at end           |
| 6  | Circular replay                    | 0=stop at end, 1=circular        |
| 7  | Make templates                     | 0=no, 1=yes                      |
| 8  | Build by code                      | 0=by shape, 1=by code            |
| 9  | Number of horizontal cursors       | 0=2, 1=4                         |
| 10 | Set which marker code to use       | 0-3 for layers 0-3               |
| 11 | Overlay drawing of traces          | 0=no overlay, 1=overlay          |
| 12 | Set size of displayed templates    | 0=small, 1=medium, 2=large       |
| 13 | Collision analysis mode            | 0=no, 1=yes: Edit WaveMark only  |
| 14 | Collision analysis use Mean width  | 0=no, 1=yes                      |
| 15 | Collision analysis Split as ideal  | 0=no, 1=yes: memory channel only |

`new%` If present, this sets the state of the item. Use 1 to select the feature (depress the button or check the box) and 0 to unselect it. Item 10 supports values 0-3.

Returns The item state at the time of the call or -1 if the item does not exist or no `item%`.

### See also:

`HCursor()`, `SSChan()`, `SSClassify()`, `SSOpen()`, `SSParam()`, `SSRun()`, `SSTempDelete()`, `SSTempGet()`, `SSTempInfo()`, `SSTempSet()`, `SSTempSizeGet()`, `SSTempSizeSet()`, `XRange()`

## SSChan()

This function sets and gets the current channel in the current spike shape dialog.

```
Func SSChan({ch%});
```

`ch%` If omitted or 0, the return value is the current channel. If greater than 0, `ch%` sets the channel and the return value is the `ch%` (or the original channel if channel `ch%` is not suitable). If the channel changes, the old channel settings are saved.

If `ch%` is -1, the return value is the number of interleaved traces in the dialog. In the New Stereotrode or New Tetrode dialogs, this is 2 or 4. In the Edit WaveMark dialog it is the number of interleaved

traces for the current channel.

If `ch%` is -2, this forces the channel configuration to be saved; the return value is 0. Use this before `FileClose()` as that does not save the current settings.

In a **New Stereotrode** or **New Tetrode** dialog `ch%` is a zero based channel index into the list of channels to combine and the return value is the channel number.

Returns The return value depends on the `ch%` argument.

**See also:**

`SSButton()`, `SSClassify()`, `SSOpen()`, `SSParam()`, `SSRun()`, `SSTempDelete()`,  
`SSTempGet()`, `SSTempInfo()`, `SSTempSet()`, `SSTempSizeGet()`, `SSTempSizeGet()`, `XRange()`

## SSClassify()

This function is equivalent to the **New Channel** button in the **New WaveMark** dialog and to the **Reclassify** button in the **Edit WaveMark** dialog. The arguments from `new%` onward are ignored in the **Edit WaveMark** dialog.

```
Func SSClassify({speed%{, new%{, type%{, noQu%}}});
```

`speed%` If this is omitted, the function returns 1 if the dialog is currently reclassifying or creating a new channel and 0 if it is not. If present, use the values:

- 1 To cancel any ongoing reclassification or writing to a new channel.
- 0 Start reclassifying or writing a new channel with display updates. This takes place in idle time, so you must provide some with `Yield()` or by using `ToolBar()`, `Interact()` or `DlgShow()`.
- 1 Reclassify or write a new channel in fast mode, with no display updates. The operation will complete before control returns to the script.

`new%` The channel number to create in the **New WaveMark** dialog.

`type%` The channel type to create. You can use types: 2 (Event-), 3(Event+), 4(Level), 5(Marker) or 6 (WaveMark) only. If omitted, type 6 (WaveMark) is used.

`noQu%` If the channel set by `new%` is already in use and this argument is omitted or 0, the user is asked if they want to overwrite it. Set non-zero for no query.

Returns If `speed%` is omitted, the return value is 1 if reclassification or writing a new channel with display updates is in progress, otherwise 0. If `speed%` is present, the return value is 1 if the action was completed, 0 if it could not be done.

**See also:**

`HCursor()`, `SSButton()`, `SSChan()`, `SSOpen()`, `SSParam()`, `SSRun()`, `SSTempDelete()`,  
`SSTempGet()`, `SSTempInfo()`, `SSTempSet()`, `SSTempSizeGet()`, `SSTempSizeGet()`, `XRange()`

## SSColApply()

This function applies the result of collision analysis to the current spike in an **Edit WaveMark** dialog. The dialog must be in **Collision analysis** mode.

```
Func SScolApply({act%});
```

`act%` This optional variable determines the action to perform. If the value is omitted it is taken as 0. The possible values are:

- 0 Test if it is possible to split the waveforms (equivalent to using the **Split spike** button in the collision analysis dialog). This can only be done if the current channel of spikes is stored in a memory channel. The return value is 1 if splitting the current spike is possible, 0 if not.
- 1 Equivalent to the **Copy codes** button in the collision analysis dialog. The return value is the number of codes that were copied. This will normally be 2 or 1, being the number of templates that contributed to the best match.
- 2 Equivalent to the **Split spike** button. The return value is -1 if it is not possible to split the current spike, otherwise the return value is the number of spikes that the current spike is split into

(typically 2, but can be 3 if two templates match and do not cross the x axis 0 point in the dialog).

Returns If the current view is an Edit WaveMark dialog, but it is not in collision analysis mode, the return value is -1. If it is any other form of WaveMark dialog, the return value is always 0. Otherwise, the return value is as described above.

### Example

The following code opens a spike shape window on the first suitable channel, changes to collision mode, then iterates through 100 spikes and copies matching codes to the second and third marker codes for all events that match 2 templates and for which the error value is less than a defined value. This assumes you have suitable templates set. The script lists the positions and matching values for all the events that satisfied the conditions. Note the use of `View(ViewLink()).Cursor(0)` to get the position of the current spike in the data file.

```
var ss%; ' handle of spike shape dialog
ss% := SSOpen(1,1); ' Open an Edit WaveMark dialog
if ss% < 0 then halt endif; ' Oops, not possible for some reason
SSButton(13,1); ' Set collision analysis mode

const minErr := 0.7; ' some arbitrary minimum error
var n%,n1%,n2%,p1,p2,err;
var i%;
var nCode% := SSButton(10, 1); ' Make code 1 the current one
for i% := 1 to 100 do
 n% := SSColInfo(err,n1%,n2%,p1,p2);
 if (n% = 2) and (err < minErr) then ' if acceptable
 SSColApply(1); ' copy codes starting at code 1
 PrintLog("%10.5f %5.2f %2d %2d %5.2f %5.2f\n",
 View(ViewLink()).Cursor(0), err, n1%, n2%, p1, p2);
 endif;
 SSRun(1); ' step to the next item
next;
SSButton(10, nCode%); ' restore the original code
```

### See also:

`SSButton()`, `SSColArea()`, `SSColInfo()`, `SSRun()`

## SSColArea()

This function gets and sets the *Important area* used for collision analysis. The command may only be used when the current view is a spike shape dialog in collision analysis mode. Note that this area is reset to lie within the template region each time the dialog enters collision analysis mode - use `SSButton(13,1)`; to do this. The command has two variants:

### Get area

**Func** `SSColArea(&start%)`

`start%` This is an integer variable, that is returned holding the zero-based offset from the start of the displayed data to the collision analysis area.

Returns The return value is the original size of the area used for collision analysis, or -1 if the spike shape dialog is not in collision analysis mode.

### Set area

**Func** `SSColArea(newSt%, newSz%);`

`newSt%` This sets the start offset of the collision analysis area.

`newSz%` This sets the size of the area (in data points) used for collision analysis.

Returns The return value is the original size of the area used for collision analysis, or -1 if the spike shape dialog is not in collision analysis mode.

Changing the area size will trigger a new matching process. Each time you enter collision analysis mode, the Important area is set to start two points after the start of the template area and end two points before the end of the template area unless this would make the area of zero size, when it is set to match the template area. In

most cases, this does what you want.

## Example

The following example increases the size of the current important area by one point in each direction

```
var start%,size%, ss%;
ss% := SSOpen(1,1); 'Open an Edit WaveMark dialog
if ss% < 0 then halt endif; 'Oops, not possible for some reason
SSButton(13,1); 'Collision analysis mode
size% := SSColArea(start%); 'get current size
SSColArea(start%-1, size%+2); 'set new size
```

### See also:

SSButton(), SSColInfo()

## SSColInfo()

This function can be used only when a Spike shape window is the current view and it is in Collision Analysis mode. The function gets information about the current state of the matching.

```
Func SSColInfo(&error{, &tmp1%{, &tmp2%{, &pos1{, &pos2}}});
```

**error** This is a real variable that is returned holding the mean error per point. This is mean ratio of the difference between the waveform and the waveform templates with respect to the template width (either point by point, or to the mean width over all templates, depending on the Mean width check box).

**tmp1%** An integer variable that is returned holding the 0-based index of the template that matches furthest to the left.

**tmp2%** An integer variable that is returned holding the 0-based index of the template that matches furthest to the right. If only one template matches, this is negative.

**pos1** A real variable that is returned holding the point position in the display at which the first point of the first template starts. This will, in general, be a fractional position as the best match is arrived at by interpolation.

**pos2** This returns the position of the second matching template in the same format as **pos1**. If there is no second template, the value is set the same as **pos1**.

**Returns** The number of matching templates (0, 1 or 2) or -1 if the spike shape dialog is not in collision analysis mode.

## Example

The following code opens a spike shape window on the first suitable channel, changes to collision mode, then iterates through 100 spikes and displays the matching information for all the ones it finds that match 2 templates. This assumes you have suitable templates set. Note the use of `View(ViewLink()).Cursor(0)` to get the position of the current spike in the data file.

```
var ss%; ' handle of spike shape dialog
ss% := SSOpen(1,1); ' Open an Edit WaveMark dialog
if ss% < 0 then halt endif; ' Oops, not possible for some reason
SSButton(13,1); ' Set collision analysis mode

var n%,n1%,n2%,p1,p2,err;
var i%;
for i% := 1 to 100 do
 n% := SSColInfo(err,n1%,n2%,p1,p2);
 if n% = 2 then ' if we match two item
 PrintLog("%10.5f %5.2f %2d %2d %5.2f %5.2f\n",
 View(ViewLink()).Cursor(0), err, n1%, n2%, p1, p2);
 endif;
 SSRun(1); ' step to the next item
next;
```

**See also:**

SSButton(), SSColArea(), SSOpen()

**SSOpen()**

This function opens and returns information about spike shape dialogs. The current view must be a time view or a spike shape dialog. Close this dialog with FileClose().

```
Func SSOpen({type%{, mode%});
```

type% This argument is assumed to be zero if omitted. Allowed values are:

- 1 The function returns the type of any open spike shape dialog associated with the current time view or 0 if there is no open dialog. Returned values are 1=Edit WaveMark, 2=New WaveMark, 3=New n-trode.
- 0 The return value is the window handle of any open spike shape dialog associated with the current view, or 0 if there is no open dialog.
- 1 Open the Edit WaveMark dialog and return its handle or 0.
- 2 Open the New WaveMark dialog and return its handle or 0.
- 3 Open the New n-trode dialog with selected channels. Return its handle or 0.

mode% If this is zero or omitted, the dialog is created invisibly. Set 1 to make it visible. This is ignored unless type% is greater than 0.

Returns The return value depends on the type% argument.

**See also:**

HCursor(), SSButton(), SSChan(), SSClassify(), SSParam(), SSRun(), ViewLink(), SSTempDelete(), SSTempGet(), SSTempInfo(), SSTempSet(), SSTempSizeGet(), SSTempSizeGet(), XRange()

**SSParam()**

This function sets the template parameters for the current spike shape dialog and reads back the state of individual parameters. This is equivalent to the template parameters dialog; see that dialog for a full description of the arguments. The command has two variants. In the first you supply all the arguments (use the value -1 to leave an argument unchanged). In the second variant you can get or set the value of a single item.

```
Func SSParam(new%,wide,rare%,AMax,PMin,flg%,mode%,nAT%,int%,tc%);
Func SSParam(item%{, value});
```

new% If new% is greater than 0, this sets number of spikes for a new template.

wide The width of a new template as a percentage of the template amplitude.

rare% Templates with spikes rarer than 1 in rate% are discounted.

AMax Maximum amplitude change for tracking or 0 for no amplitude tracking.

PMin Minimum percentage of point in the template for a possible match.

flg% This is the sum of: 1 = use minimum percentage only when building templates, 2 = Remove DC value from spike data before matching.

mode% The mode for adding spikes to the template: 0=All, 1=Autofix, 2=track.

nAT% The number of spikes for Autofix and track modes.

int% The interpolation method to use: 0=linear, 1=parabolic, 2=cubic spline.

tc% The high-pass time constant to apply to the data expressed as 2 to the power tc% data samples with tc% in the range 1 to 30. Use 31 for no high pass filter.

item% Used in the second command variant. Set -2 to copy the current settings to all channels. Set -1 to reset the values to a standard state. Set 0 to close the parameters dialog if it is open. Otherwise set 1 to 10 to select one of the arguments new% (1) to tc% (10) and the return value will be the value of that item

at the time of the call.

**value** If present, this sets the new value of the parameter when `item%` is greater than 0.

**Returns** The first command variant returns 0. The second variant, with `item%` greater than 0, returns the item value at the time of the call and `item%` less than zero returns 0. For `item%=0`, the return value is 1 if the parameter dialog was open.

**See also:**

`HCursor()`, `SSButton()`, `SSChan()`, `SSClassify()`, `SSOpen()`, `SSRun()`, `SSTempDelete()`, `SSTempGet()`, `SSTempInfo()`, `SSTempSet()`, `SSTempSizeGet()`, `SSTempSizeSet()`, `XRange()`

## SSRun()

This function gets and sets the run state of the current view, which must be a spike shape window. Use `Cursor(0)` to get the current run position. You can use `ViewLink()` to find the time view associated with the spike shape dialog.

```
Func SSRun({run%});
```

**run%** If omitted, no change is made, otherwise negative values are backwards and positive are forwards. 0 means stop, 1=step, 2=run as fast as possible, 3= run in real time, 4=- run at 91 Hz, 5=31 Hz and so on up to 15=1 Hz.

**Returns** The run state at the time of the call.

**See also:**

`HCursor()`, `SSButton()`, `SSChan()`, `SSClassify()`, `SSOpen()`, `SSParam()`, `SSTempDelete()`, `SSTempGet()`, `SSTempInfo()`, `SSTempSet()`, `SSTempSizeGet()`, `SSTempSizeSet()`, `XRange()`

## SSTempDelete()

This function deletes one or more templates for the current channel of the current view, which must be a spike shape window. If this command is used with no arguments, or with both arguments set to -1, the entire template system is cleared, equivalent to clicking the clear all templates button in the dialog.

```
Func SSTempDelete({n%, code%});
```

**n%** The zero-based index of a template or -1 for all templates. Templates that match this index and the `code%` argument are deleted. If this is omitted, -1 is used.

**code%** If this is omitted or set to -1, all templates codes are deleted. Otherwise only templates selected by `n%` that match `code%` are deleted.

**Returns** The number of deleted templates or -1 if `n%` is not a template index.

**See also:**

`SSButton()`, `SSChan()`, `SSClassify()`, `SSOpen()`, `SSParam()`, `SSRun()`, `SSTempGet()`, `SSTempInfo()`, `SSTempSet()`, `SSTempSizeGet()`, `SSTempSizeSet()`, `XRange()`

## SSTempGet()

This function gets information about the current set of templates for the current channel of the current view, which must be a spike shape window. It can also return the current raw data displayed in the template window.

```
Func SSTempGet(n%, temp[][[]]{, what%, &count%});
```

**n%** This is the zero-based index of the template to return information from or to -1 to get the last triggered waveform or -2 to get the currently displayed waveform (triggered or not). Use `SSTempSizeGet()` to get the number of displayed points and the number of points in the template. The -2 code did not exist before version 6.11.

**temp** This optional array is filled with template data. If there is insufficient data to fill it, unused entries are



unchanged. An integer or real array can be used. If the template has multiple traces, use `temp[points%][traces%]` to get real data and `temp%[points%][traces%]` to get integer data. See `ChanScale()` for an explanation of the use of integer and real data for waveform values.

**what%** Omit this or set it to 0 to return the mean template waveform. It is ignored if **n%** is 1. 1 = return the template upper limit, 2 = return the lower limit, 3 = return the template width. The width is half the distance between the upper and lower template limits.

**count%** This optional integer variable is returned set to the number of events that were accumulated into this template. It is ignored if **n%** is 1.

**Returns** If **n%** is positive, the return value is the template code or -1 if the template does not exist. If **n%** is -1, in Edit WaveMark dialogs, the return value is the currently selected sort code of the spike or -1 if you are on-line and the current data did not trigger (background data). In Create WaveMark dialogs the return value is 1 for triggered data and 0 for background data.

**See also:**

`ChanScale()`, `SSButton()`, `SSChan()`, `SSClassify()`, `SSOpen()`, `SSParam()`, `SSRun()`, `SSTempDelete()`, `SSTempInfo()`, `SSTempSet()`, `SSTempSizeGet()`, `SSTempSizeSet()`, `XRange()`

## SSTempInfo()

This function gets and sets template information for the current channel of the current view, which must be a spike shape window.

```
Func SSTempInfo({n%, item%{, val%{, noLim%}});
```

**n%** If this is omitted, the return value is the number of templates. If present, it is the zero-based index of a template or -1 for all templates (if appropriate). When returning a value, if -1 is used, the value for template 0 is returned.

**item%** This selects the information to get or set. Items 0-2 operate on all templates. Items 3-6 operate on visible templates only (**n%** in the range 0 to 19).

0 Get the index of the first template, starting at **n%**, that has a code of **val%**.

1 Get the code for template **n%**. It sets the code if **val%** is greater than 0 and less than 256.

2 Get the count of spikes added into template **n%**.

3 Get the event counter (spikes that matched the template). If **val%** is positive, it sets the event count.

4 Get the lock state of template **n%**. **val%**=0 to clear the lock and >0 to set it.

5 Change the template width by **val%** pixels. Set **val%** to 1 and -1 to mimic the increase and decrease width buttons. **noLim%** can be used.

6 Change the template width by **val%** channel units. **noLim%** can be used.

7 In collision analysis mode, get or set the exclude template check box state. **val%** is 1 to exclude, 0 to include and omitted or negative for no change. You cannot exclude all templates, at least one must be left included.

**val%** Use this argument when setting values and to locate codes when **item%** is 0.

**noLim%** Set non-zero with item codes 5 and 6 to remove the template width limits that are usually enforced to stop the template becoming too wide or too narrow. Each point of a template has a mean value and a width. It also has a minimum width, which is initialised to the original width when the template was created. The width is allowed to increase to up to 4 times the minimum width, but not to become less than the original. If you allow the width to override these limits, the minimum width is adjusted, as required.

**Returns** The value selected by **item%** at the time of the call. The return value is 0 for items 5 and 6. The return value is -1 if a template is not found.

**See also:**

`HCursor()`, `SSButton()`, `SSChan()`, `SSClassify()`, `SSOpen()`, `SSParam()`, `SSRun()`, `SSTempDelete()`, `SSTempGet()`, `SSTempSet()`, `SSTempSizeGet()`, `SSTempSizeSet()`, `XRange()`

## SSTempSet()

This function creates a template or adds the current spike in the window to a nominated template in the current channel of the current view, which must be a spike shape view. The first command variant creates a new template from user-supplied data.

The second variant adds the currently displayed data in the window to a nominated template or creates a new template from it. In the Edit WaveMark dialog this also sets the classification code of the displayed data. If this code is not in the current Marker Filter for the source channel, the displayed data changes to the next available spike. It can also renumber and sort the displayed templates and to update online templates in the 1401 to match the templates in the dialog.

```
Func SSTempSet(temp[]{[]}, code%, wide{, count%});
Func SSTempSet({n%{, code%});
```

**temp** Used when creating a new template from arrays of data. This is a one or two dimensional array holding the template shape. An integer or real array can be used. If the template has multiple traces, use `temp[points%][traces%]` for real data and `temp%[points%][traces%]` for integer data. See `ChanScale()` for an explanation of the use of integer and real data for waveform values. If the array is too short or has too few traces, missing values are set to 0.

**code%** This is used when creating a new template. Set it to 0 (or omit it in the second command variant) to use the lowest unused code or set it to the code to use for the new template in the range 1-255.

**count%** This optional argument has a default value of 1, and sets the number of spikes that contributed to the template.

**wide** A number or an array holding the template width, which is half the distance between the upper and lower template limits. If you supply an array, the shape of this array should match the `temp` array.

If you supply a number, all points are given the same width. This is in user units if `temp` is a real array and is in channel units if `temp` is an integer array.

**n%** Optional. If omitted, the currently displayed waveform is added to the best-fit template. Set it to the zero-based index of an existing template to add the current waveform to the template. Set to -1 to add a new template, in which case `code%` determines the code of the new template. Set -2 to renumber the existing templates, in which case `code%` sets the lowest numbered code used. Set -3 to sort the templates into ascending code order. Set -4 to update online templates during sampling.

**Returns** The zero-based index of the template that was added or modified or -1 if there was a problem (no matching template, unknown template index or no current waveform). The second variant with `n%` less than -1 returns 0.

### See also:

`ChanScale()`, `SSButton()`, `SSChan()`, `SSClassify()`, `SSOpen()`, `SSParam()`, `SSRun()`, `SSTempDelete()`, `SSTempGet()`, `SSTempInfo()`, `SSTempSizeGet()`, `SSTempSizeSet()`

## SSTempSizeGet()

This function gets the template size and the displayed data area for the current channel. The current view must be a spike shape window. All arguments are integer variables that are set to the template and display sizes.

```
Func SSTempSizeGet({&start%{, &show%{, &pre%}}});
```

**start%** If present, set to the point offset to the template start in the raw data display area.

**show%** If present, set to the number of data points in the raw data display area.

**pre%** If present, set to the number of pre-trigger points in the raw data display area.

**Returns** the number of data points in the template.

### See also:

`SSButton()`, `SSChan()`, `SSClassify()`, `SSOpen()`, `SSParam()`, `SSRun()`, `SSTempDelete()`, `SSTempGet()`, `SSTempInfo()`, `SSTempSet()`, `SSTempSizeSet()`

## SSTempSizeSet()

This sets the template size and the displayed data area. The template must lie within the displayed data area with at least 2 data points before and after the template. If you set a template area that exceeds these limits, the command attempts to increase the display area (if this is possible). Use `YRange()` and `Optimise()` to set the displayed y range,

```
Proc SSTempSizeSet(start%, n%, show%, pre%);
```

**start%** This is the offset to the beginning of the template from the start of the displayed area. It must be at least 2. Set -1 to leave the start point unchanged.

**n%** This sets the number of data points in the template. To keep the current number of points, either omit this argument or set it to -1.

**show%** The requested number of points to display in the main data window; it has no effect in the Edit WaveMark dialog. Omit **show%** or use -1 for no change. If odd, **show%** is reduced by 1. It is also adjusted if it is too large or too small.

**pre%** This sets the number of pre-trigger points to display in the main data window in the range 0 to **show%**-1. This has no effect in the Edit WaveMark dialog. Omit **pre%** or set -1 for no change. Out of range values are set to the appropriate limit.

### See also:

`Optimise()`, `SSButton()`, `SSChan()`, `SSClassify()`, `SSOpen()`, `SSParam()`, `SSRun()`, `SSTempDelete()`, `SSTempGet()`, `SSTempInfo()`, `SSTempSet()`, `SSTempSizeGet()`, `YRange()`

## Str\$()

This converts a number to a string.

```
Func Str$(x {,width% {,sigd%}});
```

**x** A number to be converted.

**width%** Optional minimum field width. The number is right justified in this width.

**sigd%** Optional number of significant figures in the result (default is 6) or set a negative number to set the number of decimal places as **-sigd%**.

Returns A string holding a representation of the number.

### See also:

`Asc()`, `Chr$()`, `DelStr$()`, `InStr()`, `LCase$()`, `Left$()`, `Len()`, `Mid$()`, `Print$()`, `Print()`, `Right$()`, `UCase$()`, `Val()`

## Sweeps()

This function returns the number of items accumulated in the result view.

```
Func Sweeps({set%});
```

**set%** If present, this sets the number of sweeps held by the view. You might use it to sum waveform averages in a result view so that the sweep count is correct.

Returns The value returned depends on the type of the result view (see the `SetXXXX` commands). It is an error to use this in any type of view other than a result view.

### See also:

`SetAverage()`, `SetEvtCrl()`, `SetINTH()`, `SetPhase()`, `SetPower()`, `SetPSTH()`, `SetWaveCrl()`

## System()

This function returns the operating system version as a number and gets information about desktop screens. Use the `App()` command to get the version number of Spike2.

```
Func System({get%{, scr%{, sz%[]}}});
```

- get%** A value that determines which information to return. If omitted, the value is taken as 0.
- 0 The return value is the operating system revision times 100: 351=NT 3.51, 400=95 and NT 4, 410=98, 490=Me, 500=NT 2000, 501=XP, 502=Windows Server 2003, 600=Vista, Windows Server 2008, 601=Windows Server 2008 R2, Windows 7, 602=Windows 8, Windows Server 2012. Spike2 version 8 requires Windows XP service pack 3 or later.
  - 1 The function returns information about installed desktop monitors (see `scr%` and `sz%[]`)
- scr%** Set 0 to return the number of desktop monitors; `sz%[]` gets the pixel co-ordinates of the desktop. Set to `n (>0)` to get the pixel co-ordinates of screen `n`; returns 1 for the primary monitor, 0 if not and -1 if it does not exist. Add 1000 to `scr%` to get the pixel co-ordinates of screen `n` with any areas reserved by the system (for example the taskbar) removed.
- sz%[]** Optional array of at least 4 elements to return pixel positions. Elements 0 and 1 hold the top left x and y, 2 and 3 hold bottom right x and y.

### See also:

`App()`, `System$()`, `Window()`, `WindowVisible()`

## System\$()

This returns the operating system name and accesses Spike2 environment variables. The environment holds a list of strings of the form "name=value". You can get or set the value associated with name. You can also read all the strings into a string array.

```
Func System$({var$ {,value$}});
Func System$(list$[] {,&n%});
```

- var\$** If present, this is the name of an environment variable (case insensitive).
- value\$** If present, the new value. An empty string deletes the environment variable.
- list\$** An array of strings to fill with environment strings of the form "name=value".
- n%** An optional integer that is returned holding the number of elements copied.

**Returns** With no arguments, it returns: "Windows SS build n" where SS is the operating system and n is the build number. Otherwise it returns the value of the environment variable identified by `var$` or an empty string.

Each process has its own environment. A process started with `ProgRun()` inherits a copy of the Spike2 environment, so you can pass it information. However, you cannot see environment changes made by the new process. Here are some examples of use:

```
var list$[200], value$, n%, i%;
PrintLog("%s\n", System$()); 'Print OS name
System$("fred","good"); 'Assign the value "good" to fred
PrintLog("%s\n", System$("fred")); 'get value of fred
System$("fred",""); 'Delete fred from the environment
System$(list$[], n%); 'Print all environment strings
for i%:=0 to n%-1 do PrintLog("%s\n",list$[i%]) next;
```

### See also:

`ProgRun()`, `System()`

## T

## TabSettings()

This sets and gets the tab settings for a text view. Any changes you make apply to the current view only. If you want to change the tab settings for all views, open the Edit menu preferences General tab and click the appropriate button in the Text view settings. It is possible to do this using a script with the `Profile()` command.

```
Func TabSettings({size%, flags%});
```

**size%** Tab sizes are set in units of the width of the space character in the `Default` style set for the view (style 32). Values in the range 1 to 100 set the tab size. If `size%` is 0 or omitted, no change is made. If `size%` is -1, the return value is the current `flags%` value for the text view.

**flags%** If omitted, no change is made to the flags. Otherwise, this is the sum of flag values: 1=Keep tab characters (the alternative is replace tabs with spaces), 2=show indents.

**Returns** If `size%` is positive, the return value is the tab size at the time of the call. If `size%` is -1, the return value is the `flags%` value at the time of the call.

**See also:**

`FontGet()`, `FontSet()`, `Profile()`

## TalkerReadStr()

This function controls the reading of text strings and function codes from a named talker. You could use this together with the `TalkerSendStr()` command to configure a talker before sampling (but not in a way that would cause any data it transferred to be incompatible with the target channel) or to receive commands from the talker during sampling. The original talker specification (revision 1), does not support this function. Your talker must support at least talker interface specification revision 2 to support this command. You can get information about known talkers from the `Sample` menu `Talker list` command.

The talker support in `Spike2` will queue up messages; messages are not timed out if you do not read them. There is currently a limit of 20 on the number of messages that can be queued. If you exceed this, the oldest message is lost. Each message holds a text string and a function code.

The command has three variants:

**Get the number of pending messages**

```
Func TalkerReadStr(name$);
```

**name\$** The name of the talker, which must be connected. You can see the currently connected talkers from the `Sample` menu `Talkers` command.

**Returns** The number of queued messages from the talker waiting to be read or a negative error code.

**Read the oldest message and remove it from the queue**

```
Func TalkerReadStr(name$, &msg$);
```

**name\$** The name of the talker.

**msg\$** A string variable to hold the text associated with the message. The text will be no more than 255 ASCII characters.

**Returns** A positive function code associated with the message or a negative error code.

**Discard all pending messages, get cache size**

```
Func TalkerReadStr(name$, opt%);
```

**name\$** The name of the talker.

`opt%` Set 0 to flush the queued messages and return the number discarded.  
Set to 1 to return the maximum number of queued messages or 0 if the talker is not revision 2 or not connected.

Returns The number of messages that were discarded or the maximum queue size or a negative error code.

**See also:**

`SampleTalker()`, `TalkerSendStr()`

## TalkerSendStr()

This function transmits a text string and a function code to a named talker. You could use this to configure a talker before sampling (but not in a way that would cause any data it transferred to be incompatible with the target channel) or to control the talker during sampling. The original talker specification (revision 1), does not support this function. Your talker must support at least talker interface specification revision 2 to support this command. You can get information about known talkers from the `Sample` menu `Talker list` command.

```
Func TalkerSendStr(name$, code%, msg$);
```

`name$` The name of the talker, which must be connected. You can see the currently connected talkers from the `Sample` menu `Talkers` command.

`code%` A positive 32-bit code to send; the meaning is determined by the talker. If your talker only uses strings to communicate, we recommend that you set this to 0.

`msg$` A text string to be sent to the talker; the meaning of the text is determined by the talker. If your talker only uses codes to communicate, we recommend that you set this to "". In the future we will very likely use UTF-8 coded strings, but for now we recommend that you restrict yourself to ASCII characters with codes in the range 0 to 127.

Returns 0 is returned if the the message has been passed to the talker for transmission as soon as possible. 1 if the talker is busy sending the previous message; you should try again later. Otherwise a negative error code is returned (unknown talker, not connected, timed out, talker version less than 2 so it does not accept strings).

**See also:**

`SampleTalker()`, `TalkerReadStr()`

## Tan()

This calculates the tangent of an angle in radians or converts an array of angles into tangents. Tangents of odd multiples of  $\pi/2$  are infinite, so cause computational overflow. There are  $2\pi$  radians in 360degrees.  $\pi$  is approximately 3.14159265359 ( $4.0 * \text{ATan}(1)$ ).

```
Func Tan(x|x[]{|...});
```

`x` The angle, expressed in radians, or a real array of angles. The best accuracy of the result is obtained when the angle is in the range  $-2\pi$  to  $2\pi$ .

Returns For an array, it returns a negative error code (for overflow) or 0. When the argument is not an array the function returns the tangent of the angle.

**See also:**

`Abs()`, `ATan()`, `Cos()`, `Exp()`, `Frac()`, `Ln()`, `Log()`, `Max()`, `Min()`, `Pow()`, `Rand()`, `Round()`, `Sin()`, `Sqrt()`, `Trunc()`

## Tanh()

This calculates the hyperbolic tangent of a value or an array of values.

```
Func Tanh(x|x[]{|...|});
```

**x** The value or an array of real values.

**Returns** For an array, it returns 0. Otherwise it returns the hyperbolic tangent of x.

**See also:**

`Abs()`, `ATan()`, `Cos()`, `Cosh()`, `Exp()`, `Frac()`, `Ln()`, `Log()`, `Max()`, `Min()`, `Pow()`, `Rand()`, `Round()`, `Sin()`, `Sinh()`, `Sqrt()`, `Trunc()`

## Time\$()

This function returns the current system time of day as a string. If no arguments are supplied, the returned string shows hours, minutes and seconds in a format determined by the operating system settings. To obtain the time as numbers, use `TimeDate()`. To obtain relative time and fractions of a second, use `Seconds()`.

```
Func Time$({tBase%, show%, amPm%, sep$});
```

**tBase%** Specifies the time base to show the time in. You can choose between 24 hour or 12 hour clock mode. If this argument is omitted, 0 is used.

**0** Operating system settings    **2** 12 hour format  
**1** 24 hour format

**show%** Specifies the time fields to show. Add the values of the required options together and use that as the argument. If this argument is omitted or a value of 0 is used, 7 (1+2+4) is used for 24 hour format and 15 (1+2+4+8) for 12 hour format.

**1** Show hours                                    **4** Show seconds  
**2** Show minutes                                **8** Remove leading zeros from hours

**amPm%** This sets the position of the “AM” or “PM” string in 12 hour format and has no effect in 24 hour format. If omitted, a value of zero is used. The string that gets printed (“AM” or “PM”) is specified by the operating system.

**0** Operating system settings    **2** Show to the left of the time  
**1** Show to the right of the time   **3** Hide the “AM” or “PM” string

**sep\$** This string appears between adjacent time fields. If `sep$ = “:”` then the time will appear as 12:04:45. If an empty string is entered or `sep$` is omitted, the operating system settings are used.

**See also:**

`Date$()`, `FileTime$()`, `Seconds()`, `TimeDate()`

## TimeDate()

This returns the time and date in seconds, minutes, hours, days, months, and years plus the day of the week. You can use separate variables for each field or an integer array. To get the data or time as a string, use `Date$()` or `Time$()`. To measure relative times, or times to a fraction of a second, see the `Seconds()` command. To get the current sampling time, see `MaxTime()`.

```
Proc TimeDate(&s%, &m%, &h%, &d%, &mon%, &y%, &wDay%});
Proc TimeDate(now[])
```

**s%** If `s%` is the only argument, it is set to the number of seconds since midnight. Otherwise it is set to the number of seconds since the start of the current minute.

**m%** If this is the last argument, it is set to the number of minutes since midnight. Otherwise it is set to the number of full minutes since the start of the hour.

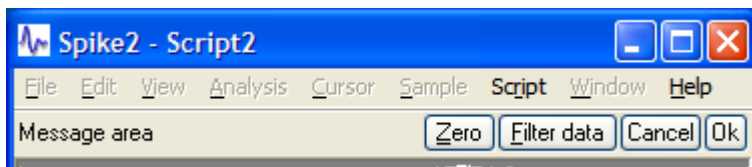
- h% If present, the number of hours since Midnight is returned in this variable.
- d% If present, the day of the month is returned as an integer in the range 1 to 31.
- mon% If present, the month number is returned as an integer in the range 1 to 12.
- y% If present, the year number is returned here. It will be an integer such as 2002.
- wDay% If present, the day of the week will be returned here as 0=Monday to 6=Sunday.
- now% [ ] If an array is the first and only argument, the first seven elements are filled with time and date data. The array can be less than seven elements long. Element 0 is set to the seconds, 1 to the minutes, 2 to the hours, and so on.

**See also:**

Date\$( ), MaxTime( ), Seconds( ), Time\$( )

## The toolbar

The toolbar is at the top of the screen, below the menu. The bar has a message area and can hold buttons that are used in the `Interact()` and `Toolbar()` commands. When you start a script, the toolbar is invisible and contains no buttons. When a script stops running, the toolbar becomes invisible (if it was visible). Do not confuse this with the system toolbar.



You can define up to 40 visible buttons, numbered from 1, in your toolbar. There is an invisible button 0, which sets an *idle* function that is called while the toolbar waits for a button to be pressed.

Buttons can be linked to the keyboard. However, any keys linked to buttons belong to the toolbar when it is active and waiting for a button press. If you link the A key to a button, each time you press A, the button is pressed, even if the text caret is in a text window.

You can also link mouse movement and mouse button presses in Time, Result and XY views to script functions and control the displayed mouse pointer.

**See also:**

Interact( ), MousePointer( ), Toolbar( ), ToolbarClear( ), ToolbarEnable( ), ToolbarMouse( ), ToolbarSet( ), ToolbarText( ), ToolbarVisible( )

## Toolbar()

This function displays the toolbar and waits for the user to click a button or press a linked key or perform a linked mouse action. If button 0 is defined with an associated *Idle* function, that function is called repeatedly while no button is pressed. If no buttons are defined or enabled, or if all buttons become undefined or disabled, the toolbar state is illegal and an error is returned. If the toolbar was not visible, it becomes visible when this command is given.

When the Toolbar is active, Spike2 ignores the `Esc` key unless an “escape button” has been set by `ToolbarSet()`.

```
Func Toolbar(text$, allow% {,help%|help$});
```

text\$ A message to display in the message area of the toolbar. The message area competes with the button area. With many buttons, the text may not be visible.

allow% A code that defines what the user can do (apart from pressing toolbar buttons). The values are the same as for `Interact()`.

1 0x0001 Can change application



|      |        |                                     |
|------|--------|-------------------------------------|
| 2    | 0x0002 | Can change the current window       |
| 4    | 0x0004 | Can move and resize windows         |
| 8    | 0x0008 | Can use File menu                   |
| 16   | 0x0010 | Can use Edit menu                   |
| 32   | 0x0020 | Can use View menu but not ReRun     |
| 64   | 0x0040 | Can use Analysis menu               |
| 128  | 0x0080 | Can use Cursor menu and add cursors |
| 256  | 0x0100 | Can use Window menu                 |
| 512  | 0x0200 | Can use Sample menu                 |
| 1024 | 0x0400 | No changes to y axis                |
| 2048 | 0x0800 | No changes to x axis                |
| 4096 | 0x1000 | No horizontal cursor channel change |

**help** This is either the number of a help item (CED internal use) or it is a help context string. This is used to set the help information that is presented when the user presses the F1 key. Set 0 to accept the default help. Set a string as displayed in the Help Index to select a help topic, for example "Cursors: Adding".

**Returns** The function returns the number of the button that was pressed to leave the toolbar, or a negative code returned by an associated button or mouse function. If the return is due to a mouse up user-defined function that returns 0, the return value will be greater than the number of any toolbar button.

The buttons are displayed in order of their item number. Undefined items leave a gap between the buttons. This effect can be used to group related buttons together.

**See also:**

`Interact()`, `ToolbarClear()`, `ToolbarEnable()`, `ToolbarMouse()`, `ToolbarSet()`, `ToolbarText()`, `ToolbarVisible()`

## ToolbarClear()

This function removes some or all of the toolbar buttons. If you delete all buttons, the `Toolbar()` function inserts an OK button, so you can get out of the `Toolbar()` function. Use `ToolbarText("")` to clear the toolbar message.

```
Proc ToolbarClear({item%});
```

**item%** If present, this is the button to clear. Buttons are numbered from 0. If omitted, all buttons are cleared. If the toolbar is visible, changes are shown immediately.

**See also:**

`Interact()`, `Toolbar()`, `ToolbarEnable()`, `ToolbarSet()`, `ToolbarText()`, `ToolbarVisible()`

## ToolbarEnable()

This function enables and disables toolbar buttons, and reports on the state of a button. Enabling an undefined button has no effect. If you disable all the buttons and then use the `Toolbar()` function, or if you disable all the buttons in a function linked to the toolbar, and there is no idle function set, a single OK button is displayed.

```
Func ToolbarEnable(item% {,state%});
```

**item%** The number of the button or -1 for all buttons. You must enable and disable button 0 with `ToolbarSet()` and `ToolbarClear()`.

**state%** If present this sets the button state. 0 disables a button, 1 enables it.

**Returns** The function returns the state of the button prior to the call, as 0 for disabled and 1 for enabled. If all buttons were selected the function returns 0. If an undefined button, or button 0 is selected, the function returns -1.

**See also:**

Interact(), Toolbar(), ToolbarClear(), ToolbarSet(), ToolbarText(),  
ToolbarVisible()

**ToolbarMouse()**

This command gives you access to the mouse positions and left button mouse clicks in Time, Result and XY views when the mouse is over a data channel while a toolbar is active. There is an example, here. The description of this command applies also to `DlgMouse()` when used with 5 or more arguments.

```
Proc ToolbarMouse(vh%, ch%, mask%, want%, Func Down%{, Func Up%{, Func
Move%});
```

**vh%** This is either the view handle of the view that you want to get mouse information from, or 0, meaning that you will accept mouse information from any suitable view.

**ch%** This is either a channel number in the view that you want mouse information for when you are over it or 0 to accept input from any channel. In an XY view, the display area is treated as belonging to channel 1, so setting 1 or 0 will work. If you set a channel number, once you have clicked on that channel, all values passed to you will be for that channel, even if you drag the mouse over a different channel. If you want to be able to click on a channel, then drag to another and be told about the other channel you must set **ch%** to 0. With **ch%** set to 0, if you drag to a place where there is no channel, you will be returned the last position that was over a channel.

**mask%** This, and the next argument (**want%**) are used when the left mouse button is clicked to decide if the script should be told about the mouse click. When the mouse button is clicked, and the conditions set here are met, the mouse becomes *owned* by the script and all mouse input will be given to the script until the mouse is released (or another application grabs the mouse). The mouse is said to be *captured*. The conditions set here are also used to decide if the script should be informed of mouse movements when the mouse is not owned by the script. Both **mask%** and **want%** are the sum of a set of values:

- 1 The left-hand mouse button is down.
- 2 The right-hand mouse button is down (releasing this button will normally display a context menu)
- 4 The `Shift` key is down
- 8 The `Ctrl` key is down
- 16 The mouse middle button is down
- 32 Extra button 1 is down (this is the left-hand side button on my mouse)
- 64 Extra button 2 is down (this is the right-hand side button on my mouse)
- 128 The `Alt` key is down
- 256 There was a left mouse button double-click

The **mask%** value determines which of these items we care about. For example if you cared about the state of the `Shift` and `Ctrl` keys, you would set the value to 12.

**want%** This argument sets the desired state of the items that you have identified with **mask%**. For example, if you only want to be told when the `Shift` key is down and the `Ctrl` key is not down, set **mask%** to 12 and **want%** to 4. Another use would be to stop the script being told when the mouse was just being moved around but had not been clicked in an area we wanted. In this case you would set **mask%** to 1 and **want%** to 1 (only tell me when the left-hand mouse button is down).

**Down%** This is the name of a user-defined function that is called when the mouse left-hand button is clicked and the conditions implied by **vh%**, **ch%**, **mask%** and **want%** are satisfied. The arguments and return value of this function are described below.

**Up%** This is the name of an optional user-defined function that is called when the mouse button is released after it has been captured. You will always get a **Down%** function call before you get an **Up%** call. If another application (rudely) takes over the mouse by popping up a window, you will also get a call to the **Up%** function. This function is described below.

**Move%** This is the name of an optional user-defined function that is called when the mouse is moved after

being captured. This function is also called when the mouse is moved when not captured by a mouse click and the conditions set by `vh%`, `ch%`, `mask%` and `want%` are satisfied. If you only want to be called during a drag operation, make sure you include the value for the left-hand mouse button in both `mask%` and `want%`.

## The user-defined mouse functions

All three functions have exactly the same arguments. The function names do not have to be `Down%`, `Up%` and `Move%`, you can choose any names that are suitable. Ideally your mouse functions (especially mouse move) should not take a long time to run; if they do, the mouse movement will feel uncomfortable and jerky. If you must trigger a more time-consuming operation, set a flag and then service it in an idle routine. The mouse pointer and any screen drawing related to the mouse is not acted upon until after your function returns, so you may get odd screen effects if you have a time-consuming function that causes screen repainting. The return value has different uses in all three cases. The functions are:

```
Func Down%(vh%, ch%, x, y, flags%);
Func Move%(vh%, ch%, x, y, flags%);
Func Up%(vh%, ch%, x, y, flags%);
```

- `vh%` The view handle of the view that the mouse is over. If you set `vh%` to a view handle value in the `ToolbarMouse()` call, then this will be that value. Your function must not close this view (the view is required to handle the return value from this function).
- `ch%` The channel number that the values of `x` and `y` relate to. If you specified a channel number in the `ToolbarMouse()` call, then this will be that value.
- `x` The x-axis value in x-axis units. If you click and drag you can get values that are outside the visible range of the x axis. If you want to scroll the view in response to this, you can do so.
- `y` The y-axis value in y axis units for the channel identified by `ch%`. If there is no y axis, the value will be 0.
- `flags%` This holds the same information as held by `mask%` and `want%`. It gives you the state of the mouse buttons and `Shift`, `Ctrl` and `Alt` keys.

## Return value

The return values have different uses for the three functions:

### **Func** Down% ( )

If you decide that you do not want to do anything with the mouse, for example the click was not over anything interesting, then return 0 and `Spike2` will decide what to do with the click. You will never get a mouse down call when the mouse is over the XY view key, or over a vertical or horizontal cursor. However, you do get priority over `Spike2` for all other clicks in a view (for sizing, for instance). Return values greater than 0 select the mouse pointer to display (this is covered below) and mean that you want to capture the mouse for script use.

You can also choose to display an indication of a selection size or area by adding a value to the return value (only 1 value can be added). If you add any of the following values and you drag beyond the left or right edges of the data area, the area will scroll (if it is allowed to).

| Value | Result                                                                                                                                                                                      |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 256   | Display a selection rectangle, as you would for zooming in and out. If you return 256, the appropriate zoom cursor (16 or 17) is selected based on the state of the <code>Ctrl</code> key). |
| 512   | Display a measurement of the distance between the start of the drag and the current position. If you return 512, the measurement cursor (19) is selected.                                   |
| 1024  | Display a line from the selection start to the current position. If you return 1024, the measurement cursor is selected.                                                                    |

### **Func** Move% ( )

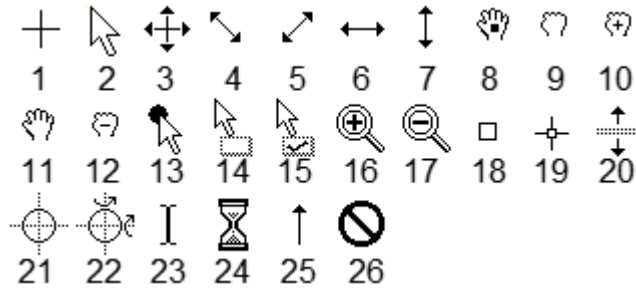
The return value from this sets the mouse pointer to be used (see below). A return value of 0 makes no change to the mouse pointer, which means that the pointer set by `Func Down% ( )` will be used for a drag operation and the cross-hair cursor (#1 in the list below) used when not dragging.

**Func Up%()**

The return value from this determines if the toolbar (or user-defined dialog for `DlgMouse()`) closes or not when the mouse button is released. Return a negative number or 0 to close the toolbar or dialog, 1 to keep running. With a dialog, a negative return acts as **Cancel** and 0 as **OK**. If the returned value closes the toolbar or dialog, the result of `Toolbar()` or `DlgShow()` is the negative returned value or a positive number greater than any button number if 0 was returned.

**Mouse pointers set by return values**

You have access to many of the mouse pointers that are available in Spike2, so you can use these to indicate that you are over an item or to show that you are dragging something. The values for the preset mouse pointers are:



The first seven and the last four of these are system mouse pointers and may have a different appearance if you have chosen a custom set of mouse pointers in the Windows system Control Panel in the Mouse section. You should be aware that 3D and animated mouse pointers can be a lot slower on some systems than simple monochrome pointers. You can also define your own mouse pointers with the `MousePointer()` command. These are assigned numbers above the range of the built-in cursors.

**Mouse double-click behaviour**

If you double-click, you will get a mouse down, possibly followed by one or more mouse moves, followed by a mouse up for the first click. The second click generates another mouse down, but this time the double-click flag will be set. It is up to you to decide if you want to undo anything that happened as a result of the initial mouse down and up. If your mouse down function returns 0, the double click will be handled by Spike2.

**Things you should not do in a user-defined mouse function**

The code in the mouse functions runs as part of the Windows code that decides which mouse pointer to display. There are a few things that you must not do inside the mouse functions:

1. Calling `FileClose()` on the view that is handling the mouse move or button notification will not close the view. This is because if it did close the view, when the mouse function returned control back to the deleted window, the program would crash.
2. Do not use up a lot of time, especially in a mouse move routine. Time consuming mouse routines will make mouse movement and button clicks feel awkward.
3. Mouse move calls after the mouse has been captured should not display user-defined dialogs or use the `Message()` or `Input()` or `Input$()` commands. If you do use these functions, you will find that the mouse is not available as it belongs to the window where the mouse was clicked. The keyboard can be used to navigate the dialog, but users will be very confused and may think that the program has crashed as mouse clicks in the dialog will have no effect.

**Debugging user-defined mouse functions**

You can set a break point in the mouse down, move and up functions. However, if you do, the function will not control the mouse pointer (as it is in use for debugging) and the mouse up function will not terminate the `Toolbar()` call if it returns 0.

**See also:**

`DlgMouse()`, `MousePointer()`, `Toolbar()`, `ToolbarMouse()` example

## ToolbarMouse() Example

This example assumes that there is a time, result or XY view open. The three mouse routines just print information to the log window. The `ToolbarMouse()` call is setup to accept any suitable window and any channel in that window. The `mask%` and `want%` arguments are set so that the mouse left-hand button must be down. This prevents the mouse move function being called unless the mouse is down.

The mouse down function requests mouse pointer 19 (the measurement pointer) and adds 1024, which causes a line to be drawn between the mouse down position and the current position.

The mouse move function returns 0, which will leave the mouse pointer unchanged. If there was no need to do anything in this routine it could be omitted. We have included it so we can print the mouse information to the log window.

The mouse up function returns 1, so that the toolbar continues to run.

```
func MouseDown%(vh%, chan%, x, y, flags%)
PrintLog("Down: chan=%2d, x=%g, y= %g, flags% = %d\n", chan%, x, y, flags%);
return 19+1024; 'cursor 19 + a line linking start to end
end;

func MouseMove%(vh%, chan%, x, y, flags%)
PrintLog("Move: chan=%2d, x=%g, y= %g, flags% = %d\n", chan%, x, y, flags%);
return 0; 'keep same cursor as for the mouse down
end;

func MouseUp%(vh%, chan%, x, y, flags%)
PrintLog("Up: chan=%2d, x=%g, y= %g, flags% = %d\n", chan%, x, y, flags%);
return 1; 'do not close the toolbar
end;

ToolbarSet(1, "Hello");
ToolbarMouse(0, 0, 1, 1, MouseDown%, MouseUp%, MouseMove%);
Toolbar("Hello", 511);
```

## ToolbarSet()

This function adds a button to the toolbar and optionally associates a function with it. When a button is added, it is added in the enabled state. You can also use this command while the toolbar is displayed to get the last button pressed (added at version 7.01). There are two command variants:

```
Func ToolbarSet(item%, label$ {,func ff%()});
Func ToolbarSet();
```

`item%` The button number in the range 1 to 40 to add or replace or 0 to set or clear a function that is called repeatedly while the toolbar waits for a button press (an *Idle* function).

You can set an “escape” key as described in `Toolbar()`, by negating `item%`. For example `ToolbarSet(-2, "Quit");` sets button 2 as the escape key.

`label$` The button label plus optional key code and tooltip as "Label|code|tip". Labels compete for space with each other; use tooltips for lengthy explanations. The label is ignored for button 0. Tooltips can be up to 79 characters long. To use a tooltip with no code use "Label||A tooltip with no code field".

To link a key to a button, place `&` before a character in the label or add a vertical bar and a key code in hexadecimal (e.g. 0x30), octal (e.g. 060) or decimal (e.g. 48) to the end of the label. Characters set by `&` are case insensitive. For example "a&Maze" generates the label aMaze and responds to m or M; the label "F1:Go|0x70" generates the label F1:Go and responds to the F1 key. Useful key codes include (nk = numeric keypad):

|                |                 |                 |           |
|----------------|-----------------|-----------------|-----------|
| 0x08 Backspace | 0x22 Page down  | 0x28 Down arrow | 0x6b nk + |
| 0x09 Tab       | 0x23 End        | 0x2e Del        | 0x6c nk , |
| 0x0d Enter     | 0x24 Home       | 0x30-39 0-9     | 0x6d nk - |
| 0x1b Escape    | 0x25 Left arrow | 0x41-5a A-Z     | 0x6e nk . |

|               |                  |                |                |
|---------------|------------------|----------------|----------------|
| 0x20 Spacebar | 0x26 Up arrow    | 0x60-69 nk 0-9 | 0x6f nk /      |
| 0x21 Page up  | 0x27 Right arrow | 0x6a nk *      | 0x70-87 F1-F24 |

Use of other keys codes or use of & before characters other than a-z, A-Z or 0-9 may cause unpredictable and undesirable effects. Code 0x6c appears on some keypads (e.g. Swedish) as the decimal separator (in place of 0x6e). Code 0x0d is also the keypad Enter code.

Beware: When the toolbar is active, it owns all keys linked to it. If A is linked, you cannot type a or A into a text window with the toolbar active.

`ff%`() This is the name of a function with no arguments. The name with no brackets is given, for example `ToolbarSet(1,"Go",DoIt%);` where `Func DoIt%()` is defined somewhere in the script. When the `Toolbar()` function is used and the user clicks on the button, the linked function runs. If the `item% 0` function is set, that function runs while no button is pressed. The function return value controls the action of `Toolbar()` after a button is pressed.

If it returns 0, the `Toolbar()` function returns to the caller, passing back the button number. If it returns a negative number, the `Toolbar()` call returns the negative number. If it returns a number greater than 0, the `Toolbar()` function does not return, but waits for the next button. An item 0 function must return a value greater than 0, otherwise `Toolbar()` will return immediately.

If this argument is omitted, there is no function linked to the button. When the user clicks on the button, the `Toolbar()` function returns the button number.

Returns The call with no arguments returns the item number of the last button pressed. This call can be used to service multiple buttons with a single user-defined function. The call that creates a button returns 0.

**See also:**

`Asc()`, `DlgButton()`, `Interact()`, `Toolbar()`, `ToolbarClear()`, `ToolbarEnable()`, `ToolbarText()`, `ToolbarVisible()`

## ToolbarText()

This replaces any message in the toolbar, and makes the toolbar visible if it is invisible. It can be used to give a progress report on the state of a script that takes a while to run.

```
Proc ToolbarText(msg$);
```

`msg$` A string to be displayed in the message area of the toolbar.

**See also:**

`Interact()`, `Toolbar()`, `ToolbarClear()`, `ToolbarEnable()`, `ToolbarSet()`, `ToolbarVisible()`

## ToolbarVisible()

This function reports on the visibility of the script toolbar, and can also show and hide it. You cannot hide the toolbar if the `Toolbar()` function is in use.

```
Func ToolbarVisible({show%});
```

`show%` If present and non-zero, the toolbar is made visible. If this is zero, and the `Toolbar()` function is not active, the toolbar is made invisible.

Returns The state of the toolbar at the time of the call. The state is returned as 2 if the toolbar is active, 1 if it is visible but inactive and 0 if it is invisible.

**See also:**

`Interact()`, `Toolbar()`, `ToolbarClear()`, `ToolbarEnable()`, `ToolbarSet()`, `ToolbarText()`

## Trim()

This function removes leading and trailing white space (spaces, tabs and end of line characters) or user-defined characters from a string variable.

```
Proc Trim(&text${, chars$});
```

`text$` The string variable to remove characters from.

`chars$` An optional list of characters to remove. If omitted, " \t\n\r" is used.

This function and the similar `TrimLeft()` and `TrimRight()` are commonly used to help parse user input that may contain multiple spaces. For example:

| Input to Trim() | After Trim() | After Trim(text\$, "1234 "); |
|-----------------|--------------|------------------------------|
| " 12AB34 "      | "12AB34"     | "AB"                         |
| " 1234 "        | "1234"       | " "                          |

### See also:

`DelStr()`, `InStr()`, `Left$()`, `Len()`, `Mid$()`, `Right$()`, `TrimLeft()`, `TrimRight()`

## TrimLeft()

This function removes leading white space (spaces, tabs and end of line characters) or user-defined characters from a string variable.

```
Proc TrimLeft(&text${, chars$});
```

`text$` The string variable to remove characters from.

`chars$` An optional list of characters to remove. If omitted, " \t\n\r" is used.

This function and the similar `Trim()` and `TrimRight()` are commonly used to help parse user input that may contain multiple spaces. For example:

| Input to TrimLeft() | After TrimLeft() | After TrimLeft(text\$, "1234 "); |
|---------------------|------------------|----------------------------------|
| " 12AB34 "          | "12AB34 "        | "AB34 "                          |
| " 1234 "            | "1234 "          | " "                              |

### See also:

`DelStr()`, `InStr()`, `Left$()`, `Len()`, `Mid$()`, `Right$()`, `Trim()`, `TrimRight()`

## TrimRight()

This function removes trailing white space (spaces, tabs and end of line characters) or user-defined characters from a string variable.

```
Proc TrimRight(&text${, chars$});
```

`text$` The string variable to remove characters from.

`chars$` An optional list of characters to remove. If omitted, " \t\n\r" is used.

This function and the similar `TrimLeft()` and `Trim()` are commonly used to help parse user input that may contain multiple spaces. For example:

| Input to TrimRight() | After TrimRight() | After TrimRight(text\$, "1234 "); |
|----------------------|-------------------|-----------------------------------|
| " 12AB34 "           | " 12AB34"         | " 12AB"                           |
| " 1234 "             | " 1234"           | " "                               |

### See also:

`DelStr()`, `InStr()`, `Left$()`, `Len()`, `Mid$()`, `Right$()`, `TrimLeft()`, `Trim()`

## Trunc()

Removes the fractional part of a real number or array. To truncate a real number to an integer, assign the real to the integer. `ArrConst()` copies a real array to an integer array.

```
Func Trunc(x|x[]{|...});
```

`x` A real number or a real array.

Returns 0 or a negative error code for an array. For a number it returns the value with the fractional part removed. `Trunc(4.7)` is 4.0; `Trunc(-4.7)` is -4.0.

**See also:**

`Abs()`, `ATan()`, `Ceil()`, `Cos()`, `Exp()`, `Floor()`, `Frac()`, `Ln()`, `Log()`, `Max()`, `Min()`, `Pow()`, `Rand()`, `Round()`, `Sin()`, `Sqrt()`, `Tan()`, `Trunc()`

## U

### U1401 commands

The U1401 commands give you direct access to the CED 1401 interface connected to your computer. You cannot use these commands while sampling unless you open a different 1401 from the one used by Spike2 for sampling. You must use the `U1401Open()` command first to take control of the 1401, and you should use the `U1401Close()` command to release the 1401 once you have finished with it. See the 1401 family programming manual for details of using the 1401; this is available from CED as part of the 1401 programming kit and also as a download from the CED web site.

**See also:**

`U1401Close()`, `U1401Ld()`, `U1401Open()`, `U1401Read()`, `U1401To1401()`, `U1401ToHost()`, `U1401Write()`

### U1401Close()

This command closes the link between the script language and a 1401 interface generated by the `U1401Open()` command. If there is no open 1401, the command is ignored.

```
Proc U1401Close();
```

**See also:**

`U1401Ld()`, `U1401Open()`, `U1401Read()`, `U1401To1401()`, `U1401ToHost()`, `U1401Write()`

### U1401Ld()

This command loads one or more 1401 commands with the option of nominating the folder to load the commands from. If no 1401 is open, the script halts.

```
Func U1401Ld(list${, path$});
```

`list$` The list of commands to load separated by commas. Include `KILL` as the first item to clear all commands first. For example: `"KILL,ADCMEM,MEMDAC"`. The file name for a command is the command name plus an extension that depends on the type of 1401. The extension is added automatically.

`path$` Optional. The path to the folder to search for the commands. If omitted, or if the command is not found in this path, the 1401 folder in the Spike2 source folder is searched, then any path indicated by the `1401DIR` environment variable, and finally, the `\1401` folder on the current drive.

Returns 0 if all commands loaded. Otherwise the bottom 16 bits is a negative error code and the upper 16 bits



is the 1-based index to the command in the list that failed to load. If  $v\%$  is the non-zero return value, the command number is  $v\%/65536$  and the error code is  $(v\% \bmod 65536 - 65536)$ .

Note that loading a command resets the state of the 1401 and stops any interrupt driven processes.

**See also:**

U1401Close(), U1401Open(), U1401Read(), U1401To1401(), U1401ToHost(), U1401Write()

## U1401Open()

This command attempts to open a 1401 for use by other U1401 commands and returns the type of the opened 1401 or a negative error code. It is not an error to call U1401Open() multiple times with no intervening U1401Close().

```
Func U1401Open({unit%});
```

unit% Optional 1401 unit number( taken as 0 if omitted) in the range 1 to 8 or 0 for the first available unit.

Returns The return value is the type of the 1401 detected: 0=standard 1401, 1=1401*plus*, 2=micro1401, 3=Power1401, 4=Micro1401 mk II, 5=Power1401 mk II, 6=Micro1401-3, 7=Power1401-3. Otherwise it is a negative error code that can be decoded by Error\$().

**See also:**

Error\$(), U1401Close(), U1401Ld(), U1401Read(), U1401To1401(), U1401ToHost(), U1401Write()

## U1401Read()

This command reads a text response from a 1401 and optionally converts it into one or more integer values or reports the number of available input lines. If no 1401 is open, the script halts. There are four variants:

```
Func U1401Read(); Get count of input lines
Func U1401Read(&text$); Read an input line as text
Func U1401Read(&v1%, &v2%, {...}); Read a line and convert to integers
Func U1401Read(arr%[]); Read a line, convert to integer array
```

text\$ A text variable returned holding the entire response.

v1% An integer variable that is returned with the first integer number read.

vn% Optional integer variables (v2% up to vn%) returned with following values. Values for which no number is returned are unchanged.

arr%[] An integer array that is filled (starting at element 0) with converted values.

Returns The version with no arguments returns the number of available input lines. The other versions return the number of items that were converted from the input text and stored to a script variable or a negative error code. If you use a variant that reads a line and there is no text to read, the command times out after about 3 seconds and returns an error code.

**See also:**

U1401Close(), U1401Ld(), U1401Open(), U1401To1401(), U1401ToHost(), U1401Write()

## U1401To1401()

This command transfers the contents of an integer array to memory in the 1401.

```
Func U1401To1401(arr%[][], addr%, size%);
```

arr% This is a one or 2 dimensional array to transfer. If you use a 2 dimensional array to interleave 4 channels of data, for example for MEMDAC, set the first dimension to 4 and the second to the number of points per channel.

addr% The start address of the block of contiguous memory in the 1401 user area to be filled with data.

`size%` Optional, the number of bytes in the 1401 that each array element is copied to. Acceptable values are 1, 2 or 4. If `size%` is omitted, 4 is used.

Returns 0 if the data transferred without a problem, or a negative error code.

**See also:**

`U1401Close()`, `U1401Ld()`, `U1401Open()`, `U1401Read()`, `U1401ToHost()`, `U1401Write()`

## U1401ToHost()

This command transfers a block of 1401 memory into an integer array.

```
Func U1401ToHost(arr%[][], addr%, size%);
```

`arr%` This is a one or 2 dimensional array to receive the data. If you use a 2 dimensional array to interleave 8 channels of data, for example for ADCMEM, set the first dimension to 8 and the second to the number of points per channel.

`addr%` The start address of the block of contiguous memory in the 1401 user area to copy data from.

`size%` Optional, taken as 4 if omitted. The number of bytes of 1401 data used to set each array element. Use 1, 2 or 4 to read 1, 2 or 4 bytes and sign extend to 32-bit integer. Use -1, -2 or -4 to read 1 or 2 or 4 bytes and zero extend to 32-bit integer.

Returns 0 if the data transferred without a problem, or a negative error code.

**See also:**

`U1401Close()`, `U1401Ld()`, `U1401Open()`, `U1401Read()`, `U1401To1401()`, `U1401Write()`

## U1401Write()

This command writes a text string to the 1401.

```
Func U1401Write(text$);
```

`text$` The text to write to the 1401. Commands to the 1401 are terminated by either a newline "\n" or a semicolon ";".

Returns 0 if the line was added to the 1401 device driver output buffer, or a negative error code.

**See also:**

`U1401Close()`, `U1401Ld()`, `U1401Open()`, `U1401Read()`, `U1401To1401()`, `U1401ToHost()`

## UCase\$()

This function converts a string into upper case. The upper-case operation may be system dependent. Some systems may provide localised uppercasing, others may only provide the minimum translation of the ASCII characters a-z to A-Z.

```
Func UCase$(text$);
```

`text$` The string to convert.

Returns An upper case version of the original string.

**See also:**

`Asc()`, `Chr$()`, `DelStr$()`, `InStr()`, `LCase$()`, `Left$()`, `Len()`, `Mid$()`, `Print$()`, `ReadStr()`, `Right$()`, `Str$()`, `Val()`

## V

**Val()**

This converts a string to a number. The converter allows the same number format as the script compiler and leading white space is ignored.

```
Func Val(text${, &nCh${, flag%}});
```

**text\$** A string that starts with a floating point number to convert. The conversion stops at the first character that is not part of the number. From version 6.10, the function will also accept a hexadecimal number if **flag%** is set to 1. In ambiguous cases, the conversion uses the format that uses the most characters of the input string, so "0xa" has the value 10, not 0 and uses all the characters. The string "0x" is converted as 0 and uses 1 character as "0x" is not a valid hexadecimal number. The expected formats are (items in curly brackets are optional, a vertical bar means use one of the characters before or after the bar):

```
{white space}{-|+}{digits}{.digits}{e|E{+|-}digits} or
{white space}0x|xhexadecimaldigits
```

**nCh%** If present, it is set to the number of characters used to construct the number.

**flag%** If present and set to 1, hexadecimal input is also acceptable.

**Returns** It returns the extracted number, or zero if no number was present. You should use the **nCh%** argument to decide if a number was processed.

**See also:**

Asc(), Chr\$(), DelStr\$(), InStr(), LCase\$(), Left\$(), Len(), Mid\$(), Print\$(), ReadStr(), Right\$(), Str\$(), UCase\$()

**VerticalMark()**

This sets up, cancels or gets information about vertical markers in the current time view. The View menu Vertical Marker command documentation has more information. Use the `FontGet()` and `FontSet()` commands to set the text font.

```
Func VerticalMark(chan%{, flags%{, just%{, hPos%{, dir%}}}});
```

**chan%** Either the channel number of an event, Marker or extended marker type to use or 0 to cancel any vertical marker or -1 to -5 to return the current values of arguments 1 (channel) to 5 (direction).

**flags%** Set this to -1 or omit for no change. Otherwise it is the sum of:

- 1 Draw on top of channel data
- 2 Use Marker code colours
- 4 Enable text for TextMark channels
- 8 No vertical line for text
- 16 No fill behind text

**just%** Sets the text justification as: 0=Left, 1=Centre, 2=Right. Use -1 for no change.

**hPos%** Sets the text baseline: 0=Below, 1=On, 2=Above the line. Use -1 for no change.

**dir%** Sets the text direction: 0=Upwards, 1=Downwards. Use -1 for no change.

**Returns** Calls with **chan%** greater than or equal to 0 return 0 for success or a negative error code. Calls with **chan%** less than 0 return the requested information.

**See also:**

FontGet(), FontSet(), Vertical markers

## View()

The `View()` function sets the current view and returns the last view handle. A view handle is a positive integer  $> 0$ . Changing the current view does not change the focus or bring the view to the front, use `FrontView()` to do that.

**Func View({vh%});**

vh% An integer argument being:

- $>0$  A valid view handle of a view that is to be made the current view. Use `ViewKind()` to test for a valid handle.
- $0$  (or omitted) no change of the current view is required.
- $<0$  If `vh%` is  $-n$ , this selects the  $n$ th duplicate of the time view that is associated with the current view. The current view can be a time, result or XY view. If the current view is a time view, this is equivalent to `Dup(n).View(-1)` in a result or XY view returns the time view from which it was created.

Returns the view handle of the view that was current at the time of the call. If an argument is passed in which is not valid, the script stops with an error.

### View().x()

The `View().x()` construction overrides the current view for the evaluation of `x()`. For example, `View(vh%).Draw(1,2)` draws view number `vh%`. It is an error if the selected view does not exist or if the function closes the original view, and the script stops.

**View(vh%).x()**

vh% A view handle of an existing view,  $0$  for the current view, or  $-n$  for the  $n$ th duplicate of the time view associated with the current view.

The equivalent code to `View(vh%).x()` is:

```
var temp%;
temp% := View(vh%); 'Save the current view
x(); 'call the user-defined or built-in function
View(temp%); 'restore the original view
```

This means that `View(vh%).FileClose()` causes an error if `vh%` is the current view.

### View().[]

The `View().[]` construction overrides the current view to give you access to the data arrays that form result view channels.

**View(vh%{,ch%}).[]**

vh% A view handle of an existing result view or  $0$  for the current result view.

ch% An optional channel number in the result view. If omitted, channel 1 is assumed.

For example:

```
ArrConst(View(0,2).[6:20], 0); 'zero 20 elements of channel 2
View(rv%).[4] := 3; 'set fifth element of channel 1
```

#### See also:

`App()`, `Dup()`, `FileClose()`, `FrontView()`, `SampleHandle()`, `ViewFind()`, `ViewKind()`, `ViewLink()`, `ViewList()`

## ViewColour()

**Deprecated.** Use `ViewColourGet()` and `ViewColourSet()`. This function gets and sets the palette colour indices of time, result and XY view items, overriding the application-wide colours set by `Colour()`. Currently you can set the background colour.

---

```
Func ViewColour(item%{, col%});
```

item% The colour item to get or set; 0=background

col% If present, the new colour index for the item. There are 40 colours in the palette with indices 0 to 39. Use -1 to revert to the application colour for the item.

Returns The palette colour index that is nearest to the item colour at the time of the call or -1 if no view colour is set. **Beware:** if you set a colour index for an item and read it back you may not get the index you set if another index holds the same colour.

**See also:**

Colour dialog, ChanColour(), Colour(), PaletteGet(), PaletteSet(), XYColour()

## ViewColourGet()

This function gets the RGB colour of a time, result or XY view item, indicating if it overrides the application-wide colours set by ColourSet().

```
Func ViewColourGet(item%{, &r, &g, &b});
```

item% The colour item to get or set; 0=background. There is only one item at present.

r g b If present, set to the red, green and blue colour values in the range 0.0 to 1.0.

Returns 1 if the returned colour overrides the application colour, 0 if it does not.

**See also:**

Colour dialog, ChanColourGet(), ColourGet(), ViewColourSet()

## ViewColourSet()

This function sets the RGB colour of time, result and XY view items, to override the application-wide colours set by ColourSet().

```
Proc ViewColourSet(item%{, r, g, b});
```

item% The colour item to get or set; 0=background. There is only one item at present.

r g b If present, These set the red, green and blue colour values in the range 0.0 to 1.0. If omitted, the item colour is set to the application default.

**See also:**

Colour dialog, ChanColourSet(), ColourSet(), ViewColourGet()

## ViewFind()

This function searches for a window with a given title and returns its view handle. This is mainly used by the script recorder to locate views that were not created as part of the recorded sequence and usually indicates code that will require editing in a finished script. Windows titles can be modified to indicate read only status and duplicated window numbers or by a script; ViewList() can be a more reliable method of locating a view that displays a known data file.

```
Func ViewFind(title$);
```

title\$ A string holding the view title to search for. The comparison is not case sensitive. If a view has been modified, the '\*' added to the displayed view title is not included in the comparison.

Returns The view handle of a view with a matching title, or 0 if no view matches.

**See also:**

WindowTitle\$, View(), ViewList()

## ViewKind()

This function returns the type of the current view or a view identified by a view handle. Types 5-7 are reserved. Type 11 windows include the sampling configuration window and control panel, the sequencer control panel and multimedia windows.

```
Func ViewKind({vh%});
```

**vh%** An optional view handle. If omitted, the type of the current view is returned. Use `-n` in a result or `XY` view to find the `n`th duplicate of the time view from which the current view was created.

Returns The type of the current view or the view identified by `vh%`. View types are:

|                   |                   |                        |                       |
|-------------------|-------------------|------------------------|-----------------------|
| -2 Invalid handle | 1 Text view       | 4 Result view          | 10 Application window |
| -1 Unknown type   | 2 Output sequence | 8 External text file   | 11 Other types        |
| 0 Time view       | 3 Script          | 9 External binary file | 12 XY view            |

**See also:**

`ChanKind()`, `SampleHandle()`, `ViewList()`

## ViewLineNumbers()

This function is used in a text-based view to display or hide line numbers and to set the number of decimal digits of space to use for a line number.

```
Func ViewLineNumbers({show%});
```

**show%** Set `-1` or omit for no change, `0` to hide line numbers, `1` to show them. Use `2-8` to set 2-8 digits of space and greater than `8` for a standard display (5 digits).

Returns `0` if line numbers were hidden at the time of the call, else the number of digits of display space allocated (2-8).

**See also:**

`Gutter()`

## ViewLink()

This function returns the view handle of the view that owns the current window. For example, you can use this to get the time view that owns a multimedia, spike shape or spike monitor window or the time view that created a result or `XY` view. This is slightly different from `view(-n)`, which finds the `n`th duplicate of the time view linked to the current time, result or `XY` view.

```
Func ViewLink();
```

Returns The handle of the linked view, or `0` if there is no such view.

This command can also be used to iterate through the process operations that are linked to the current time view. For example, if you create a new sampling document using `FileNew()` with the current sampling configuration, you may also open other result and `XY` documents and have additional channels present due to the sampling configuration. In this case, the command is:

```
Func ViewLink(n%{, mask%{, &name$}});
```

**n%** This can be `0`, meaning count the number of processes and return it, or it can be the process number to report on.

**mask%** This value determines the types of processes we wish to count or report on and is the sum of: `1` for result views generated by `SetINTH()` and similar calls, `2` for `XY` views generated by `MeasureToXY()` and `4` for data channels in the current view generated by `MeasureToChan()`. If this argument is omitted, it takes the value `3`, to report on result and `XY` views. Normally this argument will be set to `1`, `2`, `3` or `4`. There is nothing to stop you using the values `5`, `6` and `7`, but you would need to interpret the returned `name$` argument to decide if the return value was a channel number or a view

handle.

**name\$** If present, and *n%* is greater than 0, this is set to the name of the command that will process the data.

**Returns** If *n%* is 0, this returns the number of processes that match the *mask%* argument. If *n%* is greater than 0, then this returns either the handle of the view or the channel number in the current view that is the target of the process operation. If *n%* is greater than 0 and there is no corresponding process, the return value is 0.

Note that this form of the command identifies active processes, that is processes for which the `Process()` command could have some effect.

## Example

This example code lists the processes associated with the current view:

```
var vh%, n%, i%, name$;
n% := ViewLink(0, 3); 'count associated result and XY views
for i% := 1 to n% do
 vh% := ViewLink(i%, 3, name$);
 PrintLog("View Handle %d, name: %s\n", vh%, name$);
next;

n% := ViewLink(0, 4); 'count associated channels
for i% := 1 to n% do
 vh% := ViewLink(i%, 4, name$);
 PrintLog("Channel %d, name: %s\n", vh%, name$);
next;
```

### See also:

`App()`, `MMOpen()`, `SampleHandle()`, `SSOpen()`, `View()`, `ViewKind()`, `ViewList()`

## ViewList()

This function fills an integer array with a list of view handles. It never returns the view handle of the running script; use the `App()` command to get this.

**Func ViewList(list%[] {, types%});**

**list%** An integer array that is returned holding view handles. The first element of the array (element 0) is filled with the number of handles returned. If the array is too small to hold the full number, the number that will fit are returned.

**types%** The types of view to include. This is a code that can be used to filter the view handles. The filter is formed by adding the types from the list below. If this is omitted or if no types are specified for inclusion, all view handles are returned.

|   |            |     |               |      |                  |      |          |
|---|------------|-----|---------------|------|------------------|------|----------|
| 1 | Time views | 8   | Script views  | 512  | External binary  | 4096 | XY views |
| 2 | Text views | 16  | Result views  | 1024 | Application view |      |          |
| 4 | Sequencer  | 256 | External text | 2048 | Other view types |      |          |

You can also exclude views otherwise included by adding:

|       |                                                        |
|-------|--------------------------------------------------------|
| 8192  | Exclude views not directly related to the current view |
| 16384 | Exclude visible windows                                |
| 32768 | Exclude hidden windows                                 |
| 65536 | Exclude duplicates                                     |

**Returns** The number of windows that match *types%*.

The following example prints all the window titles into the log view:

```
var list%[100], i%;
ViewList(list%[]);
for i%=1 to list%[0] do
 PrintLog(view(list%[i%]).WindowTitle$()+"\n");
next;
```

**See also:**

App(), SampleHandle(), ViewKind()

## ViewMaxLines()

This function gets and optionally sets the maximum number of lines that the current text view will retain if text is added by a script. For the Log view, this value is initially set by the Edit menu Preferences in the General tab, but a script can set or clear the line limit for any text-based view. ViewStandard() clears the limit except in a Log view, where it sets the value set in the preferences.

```
Func ViewMaxLines({max%});
```

**max%** If present, this sets the maximum lines to keep after a script Print() or PrintLog() statement. If this is omitted or negative, no change is made. Set 0 for no limit and greater than zero to set a limit.

**Returns** The value of the limit at the time of the call.

Each time the line limit is exceeded, the number of lines in the file is reduced to 90% of the maximum line count by deleting the lowest numbered lines. We reduce the line count to 90% of the maximum because deleting lines from the start of the view is a slow operation (in very large views); by allowing the view to grow normally for a while this minimises the time penalty for setting this option. For example, if max% is set to 1000, the next time that a line is added and more than 1000 lines are in the file, the line count is reduced to 900. After this 100 lines can be added before the limit is tripped.

**See also:**

Edit Preferences, ViewStandard()

## ViewOverdraw()

All time views hold a list of overdraw trigger times sorted into ascending order. The use of this list is enabled by ViewTrigger(). This command adds new times to the list if overdraw mode is enabled. The command can also read back the list of trigger times. Whenever this command changes the list, the view is positioned to the last list time. It is the script equivalent of the View menu Overdraw List dialog. Trigger times are taken from a data channel, or can be supplied as times in seconds. There are two command variants:

```
Func ViewOverdraw(flag%, chan%, sTime, eTime);
```

```
Func ViewOverdraw({flag%{, time|times[]}});
```

**flag%** If -1, the times[] array must be present and times are read back from the list. Otherwise, it is the sum of the following values:

- 1 Empty the list before optionally adding any new values. If omitted, previous events are preserved unless adding new events causes them to be deleted.
- 2 Adding a time does not delete all later times in the list (Merge mode). If omitted (Normal mode), adding a time deletes all later times; the last time added becomes the current time.

**chan%** The channel number of an event, Marker, or Marker-derived channel in the time view to use as a source of trigger times. ViewTrigger(-1) returns the channel that the times in the list come from or -1 if there is a mix of channels.

**sTime** The start of the time range to search for times.

**eTime** The end of the time range to search for times.

**time** A single time to be added to the list.

**times** An array of times to be added to the list (flag% >= 0) or to hold times copied from the list (flag%=-1). Times are added in array order, so the times should be in ascending order if flag% is not 2 or 3. ViewTrigger(-1) will return -1 as there is no associated source channel.

**Returns** If flag% is -1, the return value is the number of elements of times[] that were returned holding times from the list. Otherwise, it is the resulting number of times in the list or -1 if overdrawing is disabled.



**See also:**

Overdraw list details, `ViewTrigger()`, `ViewOverdraw3D()`, Overdraw List dialog, Display Trigger dialog, Overdraw 3D dialog

## ViewOverdraw3D

This command controls 3D drawing (enabled by the `ViewTrigger()` command or the Display Trigger dialog). It is the equivalent of the View menu Overdraw 3D dialog. This command may only be used in a Time view. There are two command variants. The first sets the overdraw values, the second reads back the current settings.

```
Func ViewOverdraw3D(xProp, yProp[, xScale[, yScale[, flags%]]]);
Func ViewOverdraw3D(get%)
```

`xProp` The proportion of the available x space (in the range 0 to 1.0) to use for the 3D effect. Values outside the range 0 to 1 are limited to this range.

`yProp` The proportion of the available y space to use for the 3D effect. Values outside the range 0 to 1 are limited to this range.

`xScale` This optional argument sets how much to shrink the display width when going from the front to the back to give a perspective effect. Values are limited to the range 0 (shrink to nothing) to 1 (no shrink). If omitted, no change is made.

`yScale` How much to shrink the display height when going from the front to the back to give a perspective effect, in the range 0 to 1. Values are limited to the range 0 (shrink to nothing) to 1 (no shrink). If omitted, there is no change.

`flags%` This optional argument is omitted, no change is made. It the sum of:

- 1 Display a fixed count of frames or time range. This only has any effect if a maximum frame count or time range is set by `ViewTrigger()`. If not set, the latest frame is at the front and the oldest is at the back.
- 2 The position of a frame is determined by the frame time. If omitted, the position of a frame is set by the frame number.

`get%` The variant of the command with one argument uses this value to indicate the value to read back: -1=`xProp`, -2=`yProp`, -3=`xScale`, -4=`yScale`, -5=`flags%`

Returns 0 when setting a value or the value requested by `get%`.

**See also:**

`ViewTrigger()`, `ViewOverdraw()`, Display Trigger dialog, Overdraw List dialog, Overdraw 3D dialog

## ViewStandard()

This sets the current time, result or XY view to a standard state by making all channels and axes visible in their standard drawing mode, axis range and colour. All channels are given standard spacing and are ungrouped and the channels are sorted into the numerical order set by the Edit menu Preferences. In a time view, duplicate channels are deleted, triggered mode is disabled and any channel processing is removed. In an XY view the key is hidden and the axes are optimised.

In a text-based view it removes any maximum line limit except in the Log view, where it applies the line limit set in the Edit menu Preferences option. It also hides line numbers, displays the gutter and the folding margin (for views that support folding). All the text styles are set to the default values (equivalent to opening the FONT dialog and using Reset All) and any zooming is removed.

```
Proc ViewStandard();
```

**See also:**

`ChanOrder()`, `ChanWeight()`, `DrawMode()`, `Gutter()`, `ViewMaxLines()`, `ViewTrigger()`, `XYDrawMode()`

## ViewTrigger()

This controls the triggered drawing mode available for time views; it is a fatal error to use this command in a different view type. This command is the equivalent of the View menu Display Trigger dialog. See that for a detailed description of the arguments. The first command variant sets and enables the view trigger and clears all memory of overdrawing. The second variant enables and disables the view trigger and gets information about the current settings.

```
Func ViewTrigger(chan%, pre, hold, xZero%, cur0%, wait%, over%, col%,
 maxO%, maxT}}}}});
Func ViewTrigger({mode%});
```

- chan%** The event, marker, WaveMark, TextMark or RealMark trigger channel. Set this to 0 for a paged display on-line or during rerun.
- pre** The pre-trigger time to display, in seconds.
- hold** The minimum hold time for on-line displays, the minimum time to the next/previous trigger event for off-line use.
- xZero%** If this is set to 1, the x axis zero point (for display only) is moved to the trigger time. All time measurements are still relative to the start of the file.
- cur0%** This optional argument controls the cursor 0 action each time the view is triggered. 0=no action, 1=move cursor 0 without causing the active cursors to iterate, 2=move cursor 0 and update any active cursor positions (after a delay of *wait* seconds if this is an online trigger due to sampling a new trigger point). Set -1 for no change. If this argument is omitted, the value 0 is used.
- wait** This optional argument sets how long to wait online after a trigger before updating any active cursors when *cur0%* is 2. This allows a search that would hit the end of a file if done immediately the trigger value was detected. If this argument is omitted, the value 0 is used (for backwards compatibility). Set *wait* to -1 for no change to the current value. This argument was added at version 6.08.
- over%** Set 1 to enable overdraw, add 2 to enable 3D drawing, 0 to disable, -1 for no change. If omitted, 0 is used.
- col%** Overdrawn data colour. -1 or omitted=no change, 0=normal, 1=half intensity, 2=fade to background, 3=fade to secondary.
- maxO%** Maximum overdrawn traces from 1 to 4000, 0 for no limit or -1 or omitted for no change.
- maxT** Maximum time range of overdrawn traces up to 10000 seconds or 0 for no limit or -1 or omitted for no change.
- mode%** This command version gets information, moves to the next and previous trigger, and enables and disables the trigger. With no arguments, the command returns the enabled or disabled state as 1 or 0. If there is a single argument, it can be:
- 3 Move to the next trigger and return the trigger time or -1 if no trigger is found or if the trigger is not enabled.
  - 2 Move to the previous trigger and return the trigger time or -1 if no trigger is found or if the trigger is not enabled.
  - 1 Enable the View trigger with the current settings and return the previous trigger enable state.
  - 0 Disable the trigger and return the previous trigger enable state.
- Negative values return the current argument values:
- 1 chan%   -3 hold   -5 cur0%   -7 over%   -9 maxO%
- 2 pre   -4 xZero%   -6 wait   -8 col%   -10 maxT

**Returns** When the command is used with *mode%*, the command returns the requested information or -1 if the information is not available. All other use returns the enabled/disabled state as 1 or 0, or a negative error code.

### See also:

Display Trigger dialog, Overdraw details, DrawMode(), ViewOverdraw(), ViewOverdraw3D(), ViewStandard(), XYDrawMode()

## ViewUseColour()

This function can be used to force the display to use black and white only, or to use the colours set in the Colour dialog or by the `Colour()` command.

```
Func ViewUseColour({use%});
```

`use%` If present, a value of 0 forces Spike2 to display all windows in black and white. Any other value allows the use of colour. If omitted, no change is made.

Returns The current state as 1 if colour is in use, 0 if black and white is used.

**See also:**

`ChanColour()`, `Colour()`, `ViewColour()`, `XYColour()`

## ViewZoom()

This function is used in a text-based view to get and optionally set the zoom factor. This is the number of points to add to the nominal size of the text.

```
Func ViewZoom({zoom%});
```

`zoom%` Omit for no change or a value in the range -10 to 20 to add to the point size of all the text. The resulting minimum text size is 2 point regardless of `zoom%`.

Returns The zoom value at the time of the call.

**See also:**

`ViewStandard()`, `Zooming`

## VirtualChan()

This command controls virtual channels in the current time view. Virtual channels are defined by an algebraic expression that can include other channels (but not virtual channels). This command has the functionality of the Virtual Channel dialog. The first command variant creates and modifies a virtual channel, the second reports the state of a virtual channel.

```
Func VirtualChan(chan%, expr${, match${, binsz{, align}}});
Func VirtualChan(chan%, get${, &expr$});
```

`chan%` In the first command variant, set this to 0 to create a new virtual channel and return the channel number. The remaining arguments set the initial channel settings, otherwise default values are used. If not 0, this is the number of an existing virtual channel to modify or from which to read back the settings.

`expr$` In the first command variant, this is a string expression defining the output. See the Analysis chapter of the Spike2 manual for details. In the second command variant, this is an optional string variable that is returned holding the current expression for the channel.

`match%` This is the number of an existing waveform, RealWave or WaveMark channel to match for sample interval and alignment. If 0, the sample interval and alignment are set by the `binsz` and `align` arguments. If negative, no change is made.

`binsz` If `match%` is 0, this sets the sample interval of the channel. Values of `binsz` less than the file time resolution are ignored, as are values less than or equal to 0.

`align` If `match%` is 0, this sets the channel alignment. Values less than 0 are ignored.

`get%` In the second command variant, `get%` determines the returned value. 0 = the state of the expression parsing, 1 = `match%`, 2 = `binsz`, 3 = `align`.

Returns When creating a new channel, the return value is the new channel number or a negative error code.

When modifying an existing channel, the return value is a negative code if the `match%`, `binsz` or `align` arguments are illegal, a positive code if the expression contains an error and 0 if there was no error. The second variant returns the requested information or a negative error code. If `get%` is 0, the return value is negative if the channel is not a virtual channel, 0 if the expression is acceptable and a positive code if not.

**See also:**

More about virtual channels, `MemChan()`

**W****Window()**

This sets the position and size of the current view. Normally, positions are percentages from the top left-hand corner of the application window size. You can also set positions relative to a monitor. This can also be used to position, dock and float dockable toolbars.

```
Func Window(xLow, yLow{, xHigh{, yHigh{, scr%{, rel%}}}});
```

**xLow** Position of the left hand edge of the window in percent. When docking a dockable toolbar, the `xLow` and `xHigh` values correspond to the position of the top left corner of the window when dropped with the mouse.

**yLow** Position of the top edge of the window in percent.

**xHigh** If present, the right hand edge. If omitted the previous width is maintained. If the window is made too small, the minimum allowed size is used. If the current view is a dockable control and `yHigh` is 0, values less than 1 or greater than 4 float the window at (`xLow`, `yLow`), otherwise `xHigh` sets the docking state:

- |   |                                |   |                           |
|---|--------------------------------|---|---------------------------|
| 1 | Docked to the left window edge | 3 | Docked to the right edge  |
| 2 | Docked to the top window edge  | 4 | Docked to the bottom edge |

**yHigh** If present, the bottom edge position. If omitted the previous height is maintained. If the window is made too small, the minimum allowed size is used.

If the `Window` is dockable and `yHigh` is 0, this command sets the docked state of the window (see `xHigh`). Otherwise the window is floated with the nearest allowed width that is no more than `xHigh-xLow`. If `xHigh-xLow` is 0 or negative `yHigh` sets the height of the dockable window.

**scr%** Optional screen selector for views, dialogs and the application window. If omitted or -1, positions are relative to the application window. Otherwise, 0 selects the entire desktop rectangle and greater values select a particular screen rectangle (but see `rel%`). Add 1000 to `scr%` to get a screen rectangle with areas reserved by the system removed. See `System()` for more screen information.

**rel%** Ignored unless `scr%` is 0 or greater. Set 0 or omit for positions relative to the intersection of the rectangle set by `scr%` and the application window, 1 for positions relative to the `scr%` rectangle. If there is no intersection, there is no position change. When positioning the application window, `rel%` is treated as if it were 1.

Returns 1 if the position was valid, or -1 if the rectangle set by `scr%` and `rel%` is of zero size. Before Spike2 version 6.06 this was a `Proc` with no return value.

**Examples:**

```
View(App()).Window(0,0,100,100,0); 'Spike2 uses all desktop
View(App()).Window(0,0,100,100,2); 'Spike2 uses all second monitor
Window(0,0,100,100); 'Current view uses all application space
View(SSOpen(0)).Window(0,0,50,50); 'position a spike shape window
```

**See also:**

`App()`, `System()`, `WindowDuplicate()`, `WindowGetPos()`, `WindowSize()`, `WindowTitle$()`, `WindowVisible()`

## WindowDuplicate()

This duplicates the current time window, creating a new window that has all the settings of the current window. It does not duplicate channels; these are shared with the existing window. The new window becomes the current view and is created invisibly.

```
Func WindowDuplicate();
```

Returns The view handle of the new window, a negative error code or 0 if there are no free duplicates. There is a limit of 64 duplicates per window.

**See also:**

Window(), WindowGetPos(), WindowSize(), WindowTitle\$(), WindowVisible()

## WindowGetPos()

This gets the window position of the current view with respect to the application window, the desktop, a particular display screen or the intersection of the application window and one of these rectangles. Positions are measured from the top left-hand corner of the reference rectangle as a percentage. The `scr%` and `rel%` arguments were added at version 6.06. To get the position of a user dialog use the `DlgGetPos()` command.

```
Func WindowGetPos(&xLow, &yLow{, &xHigh, &yHigh{, scr%{, rel%}});
```

`xLow` A real variable that is set to the position of the left hand edge of the window.

`yLow` A real variable that is set to the position of the top edge of the window.

`xHigh` A real variable that is set to the position of the right hand edge of the window or that returns a docking code for a docked control bar if `yHigh` is returned as 0.

- |                                  |                             |
|----------------------------------|-----------------------------|
| 1 Docked to the left window edge | 3 Docked to the right edge  |
| 2 Docked to the top window edge  | 4 Docked to the bottom edge |

`yHigh` A real variable that is set to the position of the bottom edge of the window or to 0 if the window is docked.

`scr%` Optional screen selector for views, dialogs and the application window. If omitted or -1, positions are relative to the application window. Otherwise, 0 selects the entire desktop rectangle and greater values select a particular monitor (but see `rel%`). See `System()` for more screen information.

`rel%` Ignored unless `scr%` is 0 or greater. Set 0 or omit for positions relative to the intersection of the rectangle set by `scr%` and the application window, 1 for positions relative to the `scr%` rectangle.

Returns 1 if the position was valid, or -1 if the rectangle set by `scr%` and `rel%` is of zero size. Before Spike2 version 6.06 this was a `Proc` with no return value.

**See also:**

DlgGetPos(), System(), Window(), WindowDuplicate(), WindowSize(), WindowTitle\$(), WindowVisible()

## WindowSize()

This resizes the current view without changing the top left-hand corner position. Setting a negative size causes no change. Setting a size less than the minimum or greater than the maximum allowed sets the appropriate limit. There are no errors from this function.

```
Proc WindowSize(width, height);
```

`width` The width of the window as a percentage of the available area.

`height` The height of the window as a percentage of the available area.

You can also use this to resize the application window or a dockable control bar when it is floating; use `App()` to get the handles. For control bars, if `width` is greater than zero, its sets the width, otherwise `height` is used

to set the height. If the control bar can be resized, it will use the width or the height and will calculate the other dimension itself.

**See also:**

App(), Window(), WindowDuplicate(), WindowGetPos(), WindowTitle\$(), WindowVisible()

## WindowTitle\$()

This function gets and sets (where allowed) the current window title. Most windows can return a title. If you change a title, dependent window titles change, for example, cursor windows belonging to time views track the title of the time view.

```
Func WindowTitle$({new$});
```

**new\$** If present, this sets the new window title. Window titles must follow any system rules for length or content. Illegal titles (for example titles containing control characters) are mangled or ignored at the discretion of the system.

Returns The window title as it was prior to this call.

**See also:**

Window(), WindowDuplicate(), WindowGetPos(), WindowSize(), WindowVisible()

## WindowVisible()

This function is used to get and set the visible state of the current window. This function can also be used on the application window, however the effect will vary with the operating system.

```
Func WindowVisible({code%});
```

**code%** If present, this sets the window state. The possible states are:

- 0 Hidden, the window becomes invisible. A hidden window can be sent data, sized and so on, the result is just not visible.
- 1 Normal, the window assumes its last normal size and position and is made visible if it was invisible or iconised.
- 2 Iconised, an iconised window can be sent data, sized and so on; the result is not visible. This will dock a dockable window at its last docked position.
- 3 Maximise, make it as large as possible or float a dockable window.
- 4 Application window only; extend over all available desktop monitors.

Returns The window state prior to this call. A hidden application window may return 2, not 0.

**See also:**

FrontView(), Window(), WindowDuplicate(), WindowGetPos(), WindowSize(), WindowTitle\$()

## X

### XAxis()

This turns on and off the x axis of the current view and returns the state of the x axis.

```
Func XAxis({on%});
```

**on%** Set the axis state. If omitted, no change is made. 0=Hide axis, 1=Show axis.

Returns The axis state at the time of the call (0 or 1, as above) or a negative error code. It is an error to use this function on a view that has no concept of an x axis.

Changes made by this function do not cause a redraw immediately. The affected view is drawn at the next opportunity.

**See also:**

XAxisMode(), XAxisStyle(), XHigh(), XLow(), XRange()

## XAxisAttrib()

This function controls the choice of logarithmic or linear axis in result and XY views and control over the automatic adjustment of axis units in time, result and XY views. This command is equivalent to the controls at the bottom of the X Axis dialog. You cannot set a logarithmic axis in a time view.

**Func XAxisAttrib({flags%});**

**flags%** A value of 0 sets a linear axis with no auto-adjust of units for high zoom. Add 1 for logarithmic. Add 2 to display powers on the logarithmic axis (you must have added 1 as well for this to take effect). Add 4 to cause a linear axis to auto-adjust its units in increments of  $10^3$  at high zoom around 0, add 8 (in addition to 4) to indicate scale changes by using SI prefixes on the units. Omit this argument for no change to the attributes.

Returns The sum of the current flags set for the x-axis.

**See also:**

XAxis(), XAxisMode(), XAxisStyle(), XHigh(), XLow(), XRange(), YAxisAttrib()

## XAxisMode()

This function controls what is drawn in an x axis. This command matches some of the controls in the Show/Hide Channel dialog.

**Func XAxisMode({mode%});**

**mode%** Optional argument that controls how the axis is displayed. If omitted, no change is made. Possible values are the sum of the following:

- 1 Hide all the title information.
- 2 Hide all the unit information.
- 4 Hide small ticks on the x axis. Small ticks are hidden if big ticks are hidden.
- 8 Hide numbers on the x axis. Numbers are hidden if big ticks are hidden.
- 16 Hide the big ticks and the horizontal line that joins them.
- 32 Scale bar axis. If selected, add 4 to remove the end caps.

Returns The x axis mode value at the time of the call or a negative error code.

**See also:**

XAxis(), XAxisAttrib(), XAxisStyle(), XHigh(), XLow(), XRange(), YAxisMode()

## XAxisStyle()

This function controls the x axis display style and the major and minor tick spacing for all views that have an x axis. Tick spacing values that would cause illegible or unintelligible axes are stored but not used unless the axis range or scaling changes to make the values useful. This command matches controls in the X Axis Range dialog.

**Func XAxisStyle({style%, nTick%, major});**

**style%** In a time view, set 1 for an axis in seconds, 2 for hours minutes and seconds, 3 for time of day and 4 for milliseconds. In a result or XY view set 1 for the standard axis. If the view has units of "s" or "seconds", you can also set 4 for a milliseconds axis. Omit **style%** or use 0 to leave the style unchanged. Set -1 to return the minor tick subdivisions value, -2 to return the major tick spacing value.

`nTick%` The number of minor tick subdivisions or 0 for automatic spacing. Omit `nTick%` or set it to -1 for no change.

`major` If present, values greater than 0 sets the major tick spacing. Values less than or equal to 0 select automatic major tick spacing.

Returns If `style%` is positive or omitted it returns the style at the time of the call. See the description of `style%` for negative values.

**See also:**

`XAxis()`, `XAxisAttrib()`, `XAxisMode()`, `XHigh()`, `XLow()`, `XRange()`, `YAxisStyle()`

## XHigh()

This returns the x axis value at the right hand side of the current time, result or XY view, the end of the time range to process for a spike shape view or the line past the last fully visible line in a text view. Use with other views causes a fatal script error.

**Func XHigh()**

Returns In a time view or a spike shape dialog, the result is in seconds. In a result view, the result is in bins and can be fractional. In an XY view the result is in x axis units. It is in lines in a text view.

In a text view, the value is the first visible line number plus the number of fully visible lines. At the end of a file, the returned value can be greater than the number of lines in the file.

This example pages through a time or result view from the current position to the end.

```
while (xHigh() < MaxTime()) do Draw(XHigh()) wend;
```

**See also:**

`Draw()`, `XRange()`, `BinToX()`, `XToBin()`, `XLow()`

## XLow()

This returns the x axis value at the left hand side of the current time, result or XY view, the start of the time range to process for a spike shape view or the first visible line in a text view. It is a fatal error to use this in an inappropriate view.

**Func XLow()**

Returns In a time view or a spike shape dialog, the result is in seconds. In a result view, the result is in bins and can be fractional. In a text view, this returns a line number (the first line number in a text view is 1).

For example, this code pages a time or result view from the current position to the start of the file. In an XY view it moves the view to the left if the current position is past 0.

```
while XLow()>0 do Draw(XLow()-(XHigh()-XLow())) wend;
```

**See also:**

`Draw()`, `XRange()`, `BinToX()`, `XToBin()`, `XHigh()`

## XRange()

This sets the x axis range to display in a time result or XY view in x axis units (not bins for a result view). Unlike `Draw()`, the view does not update immediately; updates wait for the next `Draw()`, `DrawAll()`, `yield()` or some interactive activity. It also sets the time range to process in a spike shape window.

**Proc XRange(from {,to});**

`from` The left hand edge of the view in x axis units (seconds for a time view). You can set `from` to -1 in a time view that is sampling or re-running to make the view scroll automatically to show the most recent data at the right-hand edge.



`to` The right hand edge of the view. If omitted, the view stays the same width.

Values are limited to the axis range for time and result views; there is no limit in an XY view. Without `to`, it preserves the width, adjusting `from` if required. If the resulting width is less than the minimum allowed, no change is made.

**See also:**

`Draw()`, `XLow()`, `XHigh()`, `Yield()`

## XScroller()

This function gets and optionally sets the visibility of the x axis scroll bar and controls.

```
Func XScroller({show%});
```

`show%` If present, 0 hides the scroll bar and buttons, non-zero shows it.

Returns 0 if the scroll bar was hidden, 1 if it was visible.

**See also:**

`ChanShow()`, `Show/Hide channel`, `XAxisMode()`, `YAxisMode()`

## XTitle\$()

This gets and sets the x axis title in a result or XY view. In a time view, it has no effect and returns an empty string. The window updates with a new title at the next opportunity.

```
Func XTitle$({new$});
```

`New$` If present, this sets the new x axis title in a result view.

Returns The x axis title at the time of the call.

**See also:**

`ChanTitle$()`

## XToBin()

In a result view, this converts between bin numbers and x axis units; in a time view, it converts between time in seconds and the underlying Spike2 time units.

```
Func XToBin(x);
```

`x` An x axis value. If it exceeds the x axis range it is limited to the nearer end.

Returns In a result view it returns the bin position that corresponds to `x`. In a time view, it converts seconds to the underlying time units used for the file.

**See also:**

`BinToX()`, `BinSize()`

## XUnits\$()

This function gets the units of the x axis. You can also set the units in a result or XY view. The window will update with the new units at the next opportunity.

```
Func XUnits$({new$});
```

`New$` If present, this sets the new x axis units in a result view.

Returns The x axis units at the time of the call.

**See also:**

ChanUnits\$(), XTitle\$()

## XY...()

### XYAddData()

This adds data points to an XY view channel. If the axes are set to automatic expanding mode by XYDrawMode(), they will change when you add a new data point that is out of the current axis range. If the channel is set to a fixed size (see XYSize()), adding new points causes older points to be deleted once the channel is full. The first form of the command allows unrestricted x and y positions. The second form is for data that is equally spaced in the x direction.

```
Func XYAddData(chan%, x|x[]|x%[], y|y[]|y%[]);
Func XYAddData(chan%, y[], xInc{, xOff});
```

chan% A channel number in the current XY view. The first channel is number 1.

x The x co-ordinate(s) of the added data point(s). In the first form of the command, both x and y must be either single variables or arrays. If they are arrays, the number of data points added is equal to the size of the smaller array.

y The y co-ordinate(s) of the added data point(s). In the second form of the command, this is an array of equally spaced data in x.

xInc Sets the x spacing between the y data points in the second form of the command.

xOff Sets the x position of the first data point in the second form of the command. If omitted, the first position is set to 0.

Returns The number of data points which have been added successfully.

**See also:**

XYColour(), XYCount(), XYDelete(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYJoin(), XYKey(), XYRange(), XYSetChan(), XYSize(), XYSort()

### XYColour()

This gets or sets a channel line colour or a channel fill colour in the current XY view using a palette colour index.

```
Func XYColour(chan%{, col%{, item%}});
```

chan% A channel number in the current XY view. The first channel is number 1.

col% The index of the colour in the colour palette. There are 40 colours in the palette, numbered from 0 to 39. If omitted or -1, there is no colour change.

item% Set 1 for the line colour, 2 for the fill colour. Taken as 1 if omitted.

Returns The colour index in the colour palette of the nearest colour to the colour of the item at time of call or a negative error code.

This function is now deprecated. You should use ChanColourSet() and ChanColourGet() unless you need backward compatibility with older versions of Spike2.

**See also:**

Colour dialog, ChanColour(), Colour(), ViewColour(), ViewUseColour(), XYAddData(), XYCount(), XYDelete(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYJoin(), XYKey(), XYRange(), XYSetChan(), XYSize(), XYSort()

## XYCount()

This gets the number of data points in a channel in the current XY view. To find the maximum number of data points, see the `XYSize()` command.

```
Func XYCount(chan%);
```

`chan%` A channel number in the current XY view. The first channel is number 1.

Returns The number of data points in the channel or a negative error code.

### See also:

`XYAddData()`, `XYColour()`, `XYDelete()`, `XYJoin()`, `XYDrawMode()`, `XYGetData()`, `XYInCircle()`, `XYInRect()`, `XYKey()`, `XYRange()`, `XYSetChan()`, `XYSize()`, `XYSort()`

## XYDelete()

This command deletes a range of data points or all data points from one channel of the current XY view. Use `ChanDelete()` to delete the entire channel.

```
Func XYDelete(chan% {,first% {,last%}});
```

`chan%` A channel number in the current XY view. The first channel is number 1.

`first%` The zero-based index of the first point to delete. Omit to delete all points.

`last%` The zero-based index of the last data point to delete. If omitted, data points from `first%` to the last point in the channel are deleted. If `last%` is less than `first%` no data points are deleted.

Returns The function returns the number of deleted data points.

The index number of a data point depends on the current sorting method of the channel set by `XYSort()`. For different sorting methods, a data point may have different index numbers. The data points in a channel have continuous index numbers. When a point has been deleted the remaining points re-index themselves automatically.

### See also:

`ChanDelete()`, `XYAddData()`, `XYColour()`, `XYCount()`, `XYDrawMode()`, `XYGetData()`, `XYInCircle()`, `XYInRect()`, `XYJoin()`, `XYKey()`, `XYRange()`, `XYSetChan()`, `XYSize()`, `XYSort()`

## XYDrawMode()

This gets and sets the drawing and automatic axis expansion modes of a channel in the current XY view. It is an error to use this command with any other view type.

```
Func XYDrawMode(chan%, which% {,new%});
```

`chan%` A channel number in the current XY view. The first channel is number 1. This is ignored when `which%` is 5, as all XY channels share the same axes. -1 can also be used, meaning all channels.

`which%` The drawing parameter to get or set in the range 1 to 5. When setting parameters, the new value is held in the `new%` argument. The values are:

- 1 Get or set the data point draw mode. The drawing modes are:
 

|                  |             |                   |
|------------------|-------------|-------------------|
| 0 dots (default) | 3 crosses x | 6 diamonds        |
| 1 boxes          | 4 circles O | 7 horizontal line |
| 2 plus signs +   | 5 triangles | 8 vertical line   |
- 2 Get or set the size of the data points in points (units of approximately 0.353 mm). The sizes allowed are 0 to 100; 0 is invisible. The default size is 5.
- 3 Get or set the line style. If the line thickness is greater than 1 all lines are drawn as style 0. Styles are:
 

|                   |          |          |
|-------------------|----------|----------|
| 0 solid (default) | 1 dotted | 2 dashed |
|-------------------|----------|----------|

- 4 Get or set the line thickness in points. Thickness values range from 0 (invisible) to 10. The default is 1.
- 5 Get or set automatic axis range mode. This applies to the entire view, so the `chan%` argument is ignored. Values are:
  - 0 The axes do not change automatically when new data points are added.
  - 1 When new data points are added that lie outside the current x or y axis range, the data and axes screen area update at the next opportunity to display all the data.

`new%` New draw mode or axis expanding mode. If omitted, no change is made.

Returns The value of the relevant channel draw mode or axis expanding mode at the time of the call or a negative error code.

**See also:**

`XYAddData()`, `XYColour()`, `XYCount()`, `XYDelete()`, `XYGetData()`, `XYInCircle()`, `XYInRect()`, `XYJoin()`, `XYKey()`, `XYRange()`, `XYSetChan()`, `XYSize()`, `XYSort()`

## XYGetData()

This gets data points between two indices from a channel in the current XY view. It is an error to use this command with any other view type.

```
Func XYGetData(chan%, &x|x[], &y|y[] {,first% {,last%}});
```

`chan%` A channel number in the current XY view. The first channel is number 1.

`x|x[]` The returned x co-ordinate(s) of data point(s). When arrays are used, either both `x` and `y` must be arrays or neither can be. The smaller of the two arrays sets the maximum number of data points that can be returned.

`y|y[]` The returned y co-ordinate(s) of data point(s).

`first%` The zero-based index of the first data point to return. If omitted, 0 is used.

`last%` The zero-based index of the last data point returned, used when `x` and `y` are arrays. If omitted or greater than or equal to the number of data points, the final data point is the last one in the channel. If `last%` is less than `first%`, no data points are returned.

Returns The number of data points copied. If the `x` or `y` arrays are not big enough to hold all the data points from `first%` to `last%`, the return value is the array size. If `x` or `y` are not arrays, if a data point with index `first%` exists, 1 is returned.

The index number of a data point depends on the current sorting method (see `XYSort()`).

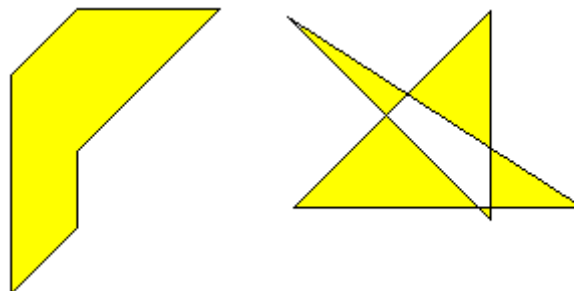
**See also:**

`XYAddData()`, `XYColour()`, `XYCount()`, `XYDelete()`, `XYDrawMode()`, `XYInCircle()`, `XYInRect()`, `XYJoin()`, `XYRange()`, `XYKey()`, `XYSetChan()`, `XYSize()`, `XYSort()`

## XYInChan()

This function returns the number of data points in a channel of the current XY view that lie inside another channel that is treated as a joined up shape. The points in the channel that defines the shape are not sorted (regardless of the sorting mode set for that channel); they are always considered in the order in which the points were added to the channel. You might use this function to let a user draw a shape (using `ToolbarMouse()`) in an XY view around data points that they want to select for a particular purpose.

For a data point to lie inside the channel, we count the number of times that a line drawn from the data point to infinity in any direction crosses a line of the channel. If it crosses an even number of times it is outside (0 is even). If it crosses an odd number of times, it is inside. This is obvious for simple shapes, but less so for complex ones. In the example picture to the right, the shaded sections are the inside and the non-shaded are the outside. Points that lie exactly on the boundary may be inside or outside; however, if you have a set of shapes that exactly tessellate to fill an area, any point placed in that area will be in only one of the shapes. If there are  $n$  points to check, and the channel that defines the shape has  $m$  points, then this algorithm takes a worst case time of  $O(n*m)$ . This means that it pays to search inside shapes with relatively few points to define them. The algorithm will be relatively quick if you are searching for points inside an area that is small compared to the area that the data points cover.



*Inside and outside*

```
Func XYInChan(chan%, shCh%{, list%[]});
```

**chan%** A channel number in the current XY view that defines the points to test against the shape.

**shCh%** The channel in the current XY view that defines the shape that the points must be inside.

**list%** An optional integer array that is returned holding the indices of the data points that were inside. If the array is too small, the function returns the number that would have been returned had the array been large enough. The index values returned are the index positions in the current sort mode for the channel. You can read the points back with `XYGetData()`.

Returns The number of data points inside the rectangle or a negative error code.

### Channel offset

This command does not make any allowance for a channel offset (`XYOffset()`). For this to work as documented, both channels must have the same offset values.

#### See also:

`XYAddData()`, `XYColour()`, `XYInCircle()`, `XYInRect()`, `XYDelete()`, `XYDrawMode()`, `XYGetData()`, `XYJoin()`, `XYKey()`, `XYRange()`, `XYSetChan()`, `XYSize()`, `XYSort()`

## XYInCircle()

This gets the number of data points inside a circle defined by  $x_c$ ,  $y_c$ , and  $r$  in the current XY view. A general point  $(x, y)$  is considered to be inside the circle if:

$$(x-x_c)^2 + (y-y_c)^2 \leq r^2$$

Points lying on the circumference are considered inside, but owing to floating-point rounding effects they may be indeterminate.

```
Func XYInCircle(chan%, xc, yc, r{, list%[]});
```

**chan%** A channel number in the current XY view. The first channel is number 1.

**xc, yc** These are the x and y co-ordinates of the centre of the circle.

**r** This is the radius of the circle.  $r$  must be  $\geq 0$ .

**list%** This is an optional integer array that is filled in with the indices of the points in the channel that were inside the circle. Points are filled from index 0 of the array until the array is full or there are no more points. The index values returned are the index positions in the current sort mode for the channel. You can read the points back with `XYGetData()`.

Returns The number of data points inside the circle or a negative error code.

#### See also:

`XYAddData()`, `XYColour()`, `XYCount()`, `XYDrawMode()`, `XYGetData()`, `XYInChan()`,

---

```
XYInRect(), XYJoin(), XYRange(), XYKey(), XYSetChan(), XYSize(), XYSort()
```

## XYInRect()

This function returns the number of data points in a channel of the current XY view that lie inside a rectangle. To be inside, a data point must lie between the low rectangle coordinate up to, but not including the high coordinate. This is so that two rectangles with a common edge will not both count a data point on the boundary.

```
Func XYInRect(chan%, xl, yl, xh, yh{, list%[]});
```

- chan%** A channel number in the current XY view.
- xl, xh** The x co-ordinates of the left and right hand edges of the rectangle. xh must be greater than or equal to xl.
- yl, yh** The y co-ordinates of the bottom and top edges of the rectangle. yh must be greater than or equal to yl.
- list%** This is an optional integer array that is filled in with the indices of the points in the channel that were inside the rectangle. Points are filled from index 0 of the array until the array is full or there are no more points. The index values returned are the index positions in the current sort mode for the channel. You can read the points back with `XYGetData()`.

Returns The number of data points inside the rectangle or a negative error code.

### See also:

`XYAddData()`, `XYColour()`, `XYInChan()`, `XYInCircle()`, `XYDelete()`, `XYDrawMode()`, `XYGetData()`, `XYJoin()`, `XYKey()`, `XYRange()`, `XYSetChan()`, `XYSize()`, `XYSort()`

## XYJoin()

This function gets or sets the data point joining method of a channel in the current XY view. Data points can be separated, joined by lines, or joined by lines with the last point connected to the first point (making a closed loop), filled or filled and framed.

```
Func XYJoin(chan% {, join%});
```

- chan%** A channel number in the current XY view. The first channel is number 1. -1 is also allowed, meaning all channels.
- join%** If present, this is the new joining method of the channel. If this is omitted, no change is made. The data point joining methods are:
- 0 Not joined by lines (this is the default joining method)
  - 1 Joined by lines. The line styles are set by `XYDrawMode()`.
  - 2 Joined by lines and the last data point is connected to the first data point to form a closed loop.
  - 3 Not joined by lines, but the channel data is filled with the channel fill colour.
  - 4 Joined by lines and filled with the channel fill colour.
  - 5 Draw as a histogram. Each consecutive pair of points (in the current sort order), defines a histogram bin. The start and end of each bin are set by the x positions and the bin contents are defined by the first point. For n bins you need n+1 points; the last point defines the end of the last bin and the y value is ignored (we recommend that you set it to 0). Drawing is most efficient when the points are ordered in increasing x order.

Returns The joining method at the time of the call or a negative error code.

### Example

The following example takes the current result view and copies it as a histogram into an XY view:

```
if (ViewKind() <> 4) then Message("Needs current result view"); halt endif;
var rv% := View(); 'Save the result view handle
var xl := BinToX(0); 'start point of first bin
var xh := BinToX(MaxTime()); 'end point of last bin
```

```

var xinc := BinSize(); 'Width of each bin
var xy% := FileNew(12, 1); 'make a visible XY view
if (xy% <= 0) then Message("Create xy view failed"); halt endif;
XYAddData(1, View(rv%,1).[], xinc, xl); 'copy data values
XYAddData(1, xh, 0); 'add right hand side of last bin
XYJoin(1, 5); 'set histogram mode
XRange(xl,xh); 'show the data range

```

**See also:**

XYAddData(), XYColour(), XYCount(), XYDelete(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYRange(), XYKey(), XYSetChan(), XYSize(), XYSort()

## XYKey()

This gets or sets the display mode and positions of the channel key for the current XY view. The key displays channel titles (set by ChanTitle\$()) and drawing symbols of all the visible channels. It can be positioned anywhere within the data area. The key can be framed or unframed, transparent or opaque and visible or invisible. ChanKey() can also be used to control XY keys.

```
Func XYKey(which%{, new});
```

which% This determines which property of the key we are interested in. Properties are:

- 1 Visibility of the key. 0 if the key is hidden (default), 1 if it is visible.
- 2 Background state. 0 for opaque (default), 1 for transparent.
- 3 Draw border. 0 for no border, 1 to draw a border (default)
- 4 Key left hand edge x position. It is measured from the left-hand edge of the x axis and is a percentage of the drawn x axis width in the range 0 to 100. The default value is 0.
- 5 Key top edge y position. It is measured from the top of the XY view as a percentage of the drawn y axis height in the range 0 to 100. The default is 0.

new If present it changes the selected property. If it is omitted, no change is made.

Returns The value selected by which% at the time of call, or a negative error code.

**See also:**

View menu Options command, ChanKey(), ChanTitle\$(), XYAddData(), XYColour(), XYCount(), XYDelete(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYJoin(), XYRange(), XYSetChan(), XYSize(), XYSort()

## XYOffset()

This command sets and gets a channel offset. The offset moves the origin of the x,y co-ordinate system when the channel is drawn. This allows you to generate a grid of histograms or to generate waterfall displays.

```
Proc XYOffset(chan%, x, y{, opt%});
Proc XYOffset(chan%, &x, &y{, opt%});
```

chan% A channel in the current XY view.

x, y Used when setting the channel offset. This is the offset in the current *axis units*.

&x, &y Used when returning the offset. Returns the offset in the current *axis units*.

opt% Omit or set to 0 to set the offset. Set -1 to return the offset.

### Axis units

In Spike2 you have the choice of drawing axes in linear or logarithmic mode. In linear mode, the axis units are the same as the user units. In logarithmic mode, the axis units are  $\text{Log}(\text{user units})$ .

This command is somewhat experimental (it was added at version 7.09) and more opt% values may be added in the future.

**See also:**  
XYRange()

## XYRange()

This function gets the range of data values of a channel or channels in the current XY view. This is equivalent to the smallest rectangle that encloses the points.

```
Func XYRange(chan%, &xLow, &yLow, &xHigh, &yHigh{, offs%});
```

- chan%** A channel number in the current XY view or -1 for all channels or -2 for all visible channels. The first channel is number 1.
- xLow** A variable returned with the smallest x value found in the channel(s).
- yLow** A variable returned with the smallest y value found in the channel(s).
- xHigh** A variable returned with the biggest x value found in the channel(s).
- yHigh** A variable returned with the biggest y value found in the channel(s).
- offs%** If this is present and non-zero, the returned values allow for any channel offset set by XYOffset(). Otherwise, any channel offset is ignored.

Returns 0 if there are no data points, or the channel does not exist, 1 if values found.

**See also:**

XYAddData(), XYColour(), XYCount(), XYDelete(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYKey(), XYSetChan(), XYSize(), XYSort()

## XYSetChan()

This function creates a new channel or modifies an existing channel in the current XY view. It is an error to use this function if the current view is not an XY view. This function can modify all properties of an existing channel without calling the XYSize(), XYSort(), XYJoin() and XYColour() commands individually.

```
Func XYSetChan(chan% {,size% {,sort% {,join% {,col%}}}});
```

- chan%** A channel number in the current XY view. If **chan%** is 0, a new channel is created. Each XY view can have maximum of 256 channels, numbered 1 to 256. The first channel is created automatically when you open a new XY view with FileNew(). If **chan%** is not 0, it must be the channel number of an existing channel to modify or -1 to modify all channels.
- size%** This sets the number of data points in the channel and how and if the number of data points can extend. The only limits on the number of data points are the available memory and the time taken to draw the view.
- A value of zero (the default) sets no limit on the number of points and the size of the channel expands as required to hold data added to it.
- If a negative size is given, for example  $-n$ , this limits the number of points in the channel to  $n$ . If more than  $n$  points are added, the oldest points are deleted to make space for the newer points. If you set a negative size for an existing channel that is smaller than the points in the channel, points are deleted.
- If a positive value is set, for example  $n$ , this allocates storage space for  $n$  data points, but the storage will grow as required to accommodate further points. Using a positive number rather than 0 can save time if you know in advance the likely number of data points as it costs time to grow the data storage.
- sort%** This sets the sorting method of data points. The sorting method is only important if the points are joined. If they are not joined, it is much more efficient to leave them unsorted as sorting a large list of points takes time. The sort methods are:
- 0 Unsorted (default). Data is drawn and sorted in the order that it was added. The most recently added point has the highest sort index.
  - 1 Sorted by x value. The index runs from points with the most negative x value to points with the most positive x value.



- 2 Sorted by y value. The index runs from points with the most negative y value to points with the most positive y value.

If this is omitted, the default value 0 is used for a new channel. For an existing channel, there is no change in sorting method.

`join%` If present, this is the new joining method of the channel. If this is omitted, no change is made to an existing channel; a new channel is given mode 0. The data point joining methods are:

- 0 Not joined by lines (this is the default joining method).
- 1 Joined by lines. The line styles are set by `XYDrawMode()`.
- 2 Joined by lines and also connect the first and last data points to form a loop.
- 3 Not joined by lines, but the channel data is filled with the channel fill colour.
- 4 Joined by lines and filled with the channel fill colour.
- 5 Draw as a histogram. See `XYJoin()` for details and an example.

`col%` If present, this sets the index of the colour in the colour palette to use for this channel. There are 40 colours with indices 0 to 39. If omitted, the colour of an existing channel is not changed. The default colour for a new channel is the colour that a user has chosen for an ADC channel in a time window. As an alternative to using a colour index, you can use `ChanColourSet()` to set the channel colour.

**Returns** The highest channel number that was affected or a negative error code. When you create a channel, the value returned is the new channel number.

**See also:**

`ChanColourSet()`, `XYAddData()`, `XYColour()`, `XYCount()`, `XYDelete()`, `XYDrawMode()`, `XYGetData()`, `XYInCircle()`, `XYInRect()`, `XYJoin()`, `XYKey()`, `XYRange()`, `XYSize()`, `XYSort()`

## XYSize()

This function gets and sets the limits on the number of data points of a channel in the current XY view. Channels can be set to have a fixed size, or to expand as more data is added. The only limit on the number of data points is the available memory and the time taken to draw data.

**Func** `XYSize(chan% {,size});`

`chan%` A channel number in the current XY view. The first channel is number 1.

`size%` This sets the number of data points in the channel and how and if the number of data points can extend. A value of zero sets no limit on the number of points and the size of the channel expands as required to hold data added to it.

If a negative size is given, for example `-n`, this limits the number of points in the channel to `n`. If more than `n` points are added, the oldest points are deleted to make space for the newer points. If you set a negative size for an existing channel that is smaller than the points in the channel, points are deleted.

If a positive value is set, for example `n`, this allocates storage space for `n` data points, but the storage will grow as required to accommodate further points. Using a positive number rather than 0 can save time if you know in advance the likely number of data points as it costs time to grow the data storage.

If this is omitted, there is no change to the size.

**Returns** If the number of points for the channel is fixed at `n` points, the function returns `n`. Otherwise, the function returns the maximum number of points that could be stored in the channel without allocating additional storage space.

**See also:**

`XYAddData()`, `XYColour()`, `XYCount()`, `XYDelete()`, `XYDrawMode()`, `XYGetData()`, `XYInCircle()`, `XYInRect()`, `XYJoin()`, `XYKey()`, `XYRange()`, `XYSetChan()`, `XYSort()`

## XYSort()

In the current XY view, gets or sets the sorting method of the channel.

```
Func XYSort(chan% {,sort%});
```

**chan%** the channel number in the current XY view. The first channel is number 1.

**sort%** This sets the sorting method of data points. The sorting method is only important if the points are joined. If they are not joined, it is much more efficient to leave them unsorted as sorting a large list of points takes time. The sort methods are:

- 0 Unsorted (default). Data is drawn and sorted in the order that it was added. The most recently added point has the highest sort index. This is the fastest (most efficient) sorting method.
- 1 Sorted by x value. The index runs from points with the most negative x value to points with the most positive x value.
- 2 Sorted by y value. The index runs from points with the most negative y value to points with the most positive y value.

If this is omitted, there is no change in sorting method.

**Returns** The function returns the sorting method at time of call or a negative error code.

**See also:**

XYAddData(), XYColour(), XYCount(), XYDelete(), XYDrawMode(), XYGetData(), XYInCircle(), XYInRect(), XYJoin(), XYKey(), XYRange(), XYSetChan(), XYSize()

## Y

### YAxis()

This function is used to turn the y axes on and off, in the current view and to find the state of the y axes in a view.

```
Func YAxis({on%});
```

**on%** Optional argument that sets the state of the axes. If omitted, no change is made. Possible values are:

- 0 Hide all y axes in the view.
- 1 Show all y axes in the view.

**Returns** The state of the y axes at the time of the call (0 or 1) or a negative error code.

**See also:**

ChanOffset(), ChanScale(), Grid(), Optimise(), XAxis(), XScroller(), YAxisMode(), YHigh(), YLow(), YRange()

### YAxisAttrib()

This function controls the options available through check boxes in the Y Axis dialog for all views that have y-axes. If you set values that would cause illegible or unintelligible axes, they are stored but not used unless other axis attributes change to make the values useful. You cannot set logarithmic axes in a time view.

```
Func YAxisAttrib(cSpc{, flags%});
```

**cSpc** A channel specifier or -1 for all, -2 for visible and -3 for selected channels. When multiple channels are specified, returned values are for the first channel.

**flags%** A value of 0 sets a linear axis with no auto-adjust of units for high zoom. Add 1 for logarithmic. Add 2 to display powers on the logarithmic axis (you must have added 1 as well for this to take effect). Add 4 to cause a linear axis to auto-adjust its units by powers of  $10^3$  at high zoom around 0. Add 8 (in addition to 4) to use an SI prefix on the units to indicate scales. Omit this argument for no change to the attributes.

Returns The sum of the current flags set for the y-axis of the first channel in the channel specification. If the channel does not exist, the result is 0.

**See also:**

`YAxis()`, `YAxisMode()`, `YAxisStyle()`, `YHigh()`, `YLow()`, `YRange()`, `XAxisAttrib()`

## YAxisLock()

This function locks and unlocks the axes of grouped channels and reports on the locked state of grouped channels. If you lock a group, the grouped channels keep their own axis ranges, but display using the axis of the first channel in the group. The `YRange()`, `YHigh()` and `YLow()` commands operate on the information stored with a channel. To change the displayed range of grouped and locked channels, you must use `YRange()` on the first channel in a group.

```
Func YAxisLock(chan%{, opt%{, vOffs}});
```

`chan%` A channel that is in the group that you wish to address.

`opt%` If present, values of 1 and 0 set and unset the locked state. A value of -1 returns the visual offset per channel for the group. If omitted, no change is made.

`vOffs` If present, this sets the y axis display offset to apply between channels in the group. The *n*th channel has a visual offset of  $(n-1)*offs$ .

Returns The current locked state of the group unless `opt%` was -1, when it returns the y axis visual offset per channel for the group.

**See also:**

`ChanOrder()`, `YAxisMode()`, `YHigh()`, `YLow()`, `YRange()`

## YAxisMode()

This function controls what is drawn in a y axis and where the y axis is placed with respect to the data.

```
Func YAxisMode({mode%{, hal%{, hsp%{, hvp%}}});
```

`mode%` Optional argument that controls how the axis is displayed. If omitted or negative, no change is made. Positive values are the sum of:

- 1 Hide all the title information.
- 2 Hide all the unit information.
- 4 Hide y axis small ticks. They are also hidden when big ticks are hidden.
- 8 Hide y axis numbers. They are also hidden when big ticks are hidden.
- 16 Hide the big ticks and the vertical line that joins them.
- 32 Scale bar axis. If selected add 4 to remove the end caps.
- 4096 Place the y axis on the right of the data.
- 8192 Horizontal text for the title and units.

`hal%` The horizontal label alignment as -1 or omitted for no change, 0 for centred, 1 for aligned to the edge.

`hsp%` Set the horizontal label character space in the range 2-17 or -1 or omitted for no change.

`hvp%` Set the horizontal label vertical position as 0 for centred, 1 for top, 2 for bottom or -1 or omitted for no change.

Returns If `mode%` is positive, omitted or -1 it returns the `mode%` value at the time of the call or a negative error code. Set `mode%` to -2 to return the current value of `hal%`, -3 for `hsp%` or -4 for `hvp%`.

**See also:**

`ChanNumbers()`, `YAxis()`, `YAxisStyle()`, `YHigh()`, `YLow()`, `YRange()`

## YAxisStyle()

This function controls the y axis major and minor tick spacing. If you set values that would cause illegible or unintelligible axes, they are stored but not used unless the axis range or scaling changes to make the values useful.

```
Func YAxisStyle(cSpC, opt%{, major});
```

**cSpC** A channel specifier or -1 for all, -2 for visible and -3 for selected channels. When multiple channels are specified, returned values are for the first channel.

**opt%** Values greater than 0 set the number of subdivisions between major ticks. 0 sets automatic small tick calculation. Use -1 for no change. Values less than -1 return information, but do not change the axis style. The maximum number of subdivisions is 25.

**major** If present and **opt%** is greater than -2, values greater than 0 sets the major tick spacing. Values less than or equal to 0 select automatic major tick spacing.

**Returns** If **opt%** is -2 this returns the current number of forced subdivisions or 0 if they are not forced. If **opt%** is -3 this returns the current major tick spacing if forced or 0 if not forced. Otherwise the return value is 0 or a negative error code. If multiple channels are specified the return value is for the first channel in the list.

**See also:**

Channel specifiers, YAxis(), YAxisMode(), YHigh(), YLow(), YRange(), XAxisStyle()

## YHigh()

This function returns the upper y axis limit for the y axis of a channel in a time, result or XY view. The return value ignores the grouped or locked state of the channel.

```
Func YHigh(chan%);
```

**chan%** The channel number (use 0 for an XY view).

**Returns** The value at the appropriate end of the axis.

**See also:**

YAxisLock(), YLow(), YRange(), ChanScale(), ChanOffset(), Optimise()

## Yield()

This function suspends script operation for a user defined time and allows the system to idle. During the idle time, invalid screen areas update, you can interact with the program and Spike2 has the opportunity to do housekeeping such as releasing temporary system resources. If your script runs for long periods without using `Interact()` or `Toolbar()`, adding an occasional `Yield()` can make it feel more responsive and stop the system marking Spike2 as "not responding".

```
Func Yield({wait{, allow%}});
```

**wait** The minimum time to wait, in seconds. If omitted or 0, Spike2 gives the system one idle cycle. If greater than 0 and there is still waiting time left after an idle cycle completes, other processes are given the opportunity to run between idle cycles. If set negative, there is no idle cycle but the **allow%** argument is applied.

**allow%** This defines what the user can do during the wait period. See `Interact()` for the allowed values. The **allow%** value is cancelled after the command unless **wait** is 0 or negative.

**Returns** The function returns 1. We may add more return codes in future versions.

**See also:**

Interact(), MaxTime(), Seconds(), Toolbar(), YieldSystem()

## YieldSystem()

The `YieldSystem()` command causes Spike2 to surrender the current time slice and suspends the user interface and script thread for a user-defined time or until a new message arrives in the Spike2 input queue. It has no effect on sampling, which runs in a separate thread. Unlike `Yield()`, it does not allow Spike2 to idle.

To share the system CPU(s) among competing processes, the operating system allocates time slices of around 10 milliseconds based on process priorities and recent process activity. A process can surrender a time slice if it has nothing to do. A typical application spends most of its time waiting for user input, which appears as messages in the application message queue; it will surrender a time slice if the message queue is empty unless it has a task to work on. Spike2 normally surrenders time slices, but if you run a script, it runs for the full time slice unless it is in `Yield()`, `Interact()`, or you use `ToolBar()` or `DlgShow()` without an idle routine.

```
Proc YieldSystem({wait});
```

`wait` The time to wait, in seconds, before resuming the thread. Values are rounded to the nearest millisecond. Values greater than 10 are treated as 10 seconds; values less than -10 are treated as -10 seconds.

For `wait` values greater than 0, the wait is ended by unserved messages; keyboard and mouse activity, timers for screen updates and the like cause messages. If `wait` is 0 or omitted, the current time slice is surrendered, but if Spike2 is the highest priority task it will be re-scheduled immediately. Negative values suspend Spike2 for `-wait` seconds regardless of messages.

`YieldSystem()` with `wait` values greater than 0 returns immediately if there are messages in the input queue. Unless you allow Spike2 to idle, either with a `Yield()` call or with `ToolBar()` or `DlgShow()`, there will always be pending messages, so it will have no effect. If you have a script loop that causes 100% CPU usage, inserting:

```
Yield();YieldSystem(0.001);
```

will give other processes a chance to run. Increasing the wait time up to 0.05 will further reduce the CPU usage. Larger values have little additional effect due to timer messages ending the wait early. To give as much system time as possible to other tasks without allowing Spike2 to idle, you can use:

```
YieldSystem(-0.001);
```

In this case, increasing the time to -10.0 will have an effect; Spike2 will feel completely unresponsive until the time period has elapsed.

### See also:

`Interact()`, `DlgShow()`, `Seconds()`, `ToolBar()`, `Yield()`

## YLow()

This function returns the lower y axis limit for the y axis of a channel in a time, result or XY view. The return value ignores the grouped or locked state of the channel.

```
Func YLow(chan%);
```

`chan%` The channel number. Use 0 for an XY view.

Returns The value at the appropriate end of the axis.

### See also:

`YAxisLock()`, `YHigh()`, `YRange()`, `ChanScale()`, `ChanOffset()`, `Optimise()`

## YRange()

This sets the displayed y axis range for a channel in a Time, Result or XY view. It can also be used in a Spike Shape dialog. It has no effect if the current channel display mode has no y axis. If the y range changes, the display updates at the next opportunity. If you omit `low` and `high`, Spike2 displays the “full” range of the channel, equivalent to the Show All button in the Y Axis dialog. In a Spike Shape dialog, the value of `low` or

high farthest from zero sets the displayed range.

```
Proc YRange(cSpc{, low, high});
```

- cSpc** A channel specifier for the channels or -1 for all, -2 for visible or -3 for selected. This is ignored in a Spike Shape dialog and should be set to 0.
- low** The value for the bottom of the y axis. If omitted, and the channel type has a known range, Spike2 sets `low` and `high` to suitable limits. For example, for a waveform channel displayed as a waveform the limits are those set by the 16-bit nature of the data. For a sonogram channel, the limits are 0 and half the sampling rate. For an event channel the limits are 0 and the maximum sustained event rate set in the sampling configuration dialog.
- high** The value for the top of the y axis. If `low` and `high` are the same, or too close, the range is not changed. It is an error to supply `low` and omit `high`.

**See also:**

Channel specifiers, `YAxisLock()`, `YHigh()`, `YHigh()`, `YLow()`, `ChanScale()`, `ChanOffset()`, `Optimise()`

## Z

### ZeroFind()

Find a zero (root) of a user-defined continuous function using Brent's method. You must supply a range to search that contains a zero. The method is iterative with each iteration reducing the search range. Iterations stop when a zero is found, the search range becomes smaller than a supplied tolerance or an iteration limit is reached. The need to supply a range that brackets a zero means that you must have some understanding of the general shape of the function before using `ZeroFind()`.

```
Func ZeroFind(&x, f(x), a, b{, tol{, maxIt%}});
```

- x** A real variable that is returned with a value that is within `tol` of a position for which `f(x)` is zero.
- f** The name (no brackets or argument) of a user-defined function that takes a single real argument and that returns a real value.
- a,b** These values define a range to be searched for a zero. `f(a)` and `f(b)` must both be non-zero and must have different signs.
- tol** This specifies how close the returned `x` must be to the exact result and must be positive. If omitted or zero, the tolerance is set based on the root position and floating point precision.
- MaxIt%** The maximum number of iterations of the algorithm in the range 1 to 200. If omitted, a maximum of 100 iterations are set.
- Return** The number of iterations left after the tolerance is achieved, or 0 if it was not, or -1 if the initial range (`a, b`) does not include a zero.

### Examples

The following example calculates the roots of the equation  $x^2-2=0$ .

```
Func root2(x) return x*x-2 end;
var r1,r2;
ZeroFind(r1, root2, 0, 2); 'Find a root between 0 and 2
ZeroFind(r2, root2, 0, -2); 'Find a root between 0 and -2
printlog("Roots at %g and %g\n", r1, r2);
```

The result of this example could have been obtained by simple algebra; the roots are obviously  $\pm\sqrt{2}$ . However, in some cases the function may not be analytic. For example, suppose we want to demonstrate the often quoted result that 95% of normally distributed values lie within two standard deviations of the mean. The indefinite integral of a Gaussian with unit standard deviation can be obtained with:

```
Func NormProp(x) return GammaP(0.5, 0.5*x*x); end;
```

This returns the fraction of the data that lies within  $x$  standard deviations of the mean of a normal distribution. We want the value of  $x$  for which  $\text{NormProp}(x) = 0.95$ . Put another way, we want  $\text{NormProp}(x) - 0.95$  to be zero. The following script does this:

```
Func NormProp(x) return GammaP(0.5, 0.5*x*x); end;
Func Root(x) return NormProp(x)-0.95 end;

var sd;
ZeroFind(sd, Root, 0, 5); 'get 95% level
printlog("95%% at %4.2f\n", sd);
```

You can use this function in some circumstances to calculate the inverse of a function when no other method is available. That is, if  $y = f(x)$ , then  $x = f^{-1}(y)$ . For example, in the above case we were finding the inverse of the  $\text{NormProp}(x)$  function at the specific value 0.95. To tabulate the function for a range of values we would change the `Root()` function to:

```
var prop;
Func Root(x) return NormProp(x)-prop end;
```

Now, by setting `prop` to a value in the range between 0 and 1 (but not to 0 and 1), we can map the inverse function:

```
for prop := 0.01 to 0.99 step 0.01 do
 ZeroFind(sd, Root, 0, 100);
 printlog("%4.2f %5.3f\n", prop, sd);
next;
```

We have omitted 0 and 1 from our range because we cannot set a range that meets the criteria for the arguments `a` and `b` for these values (the result from `NormProp()` can only lie in the range 0 to 1, and only reaches 1 at infinity).

**See also:**

Normal distribution CDF

## Curve fitting

### Introduction

It frequently happens that you have a set of data values  $(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)$  that you wish to test against a theoretical model  $y = f(x, \mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \dots)$  where the  $\mathbf{a}_i$  are coefficients that are to be set to constant values which give the *best fit* of the model to the data values.

For example, if we were looking at the extension of a spring ( $y$ ) as it is loaded by weights ( $x$ ), we might wish to fit the straight line  $y = \mathbf{a}_0 + \mathbf{a}_1 x$  to some measured data points so that we could measure a weight by the extension it caused. A careful experimenter might also wish to know what the probable error was in  $\mathbf{a}_0$  and  $\mathbf{a}_1$ , so that the probable error in any weight deduced from an extension would be known. An even more cautious experimenter might want to know if the straight-line formula was likely to model the measured data.

To avoid repeating definitions throughout the remainder of this chapter the following will be taken as defined. We apologise to the statisticians who may read this and shudder.

#### mean

Given a set of  $n$  values  $y_j$ , the mean is the sum of the  $n$  values divided by  $n$ .

#### variance

If the mean of a set of  $n$  data values  $y_j$  is  $y_m$ , then the variance (sigma squared) of this set of values is: the sum of the squares of  $y_j - y_m$  divided by  $n$  if  $y_m$  is known independently of the data values and divided by  $(n - 1)$  if  $y_m$  is calculated from the data values  $y_j$ . For a data set of any reasonable size, the use of  $n-1$  or  $n$  in the denominator should make little difference.

#### standard deviation

The standard deviation (sigma) of a data set is the square root of the variance. Both the variance and the standard deviation are used as measures of the width of the distribution.

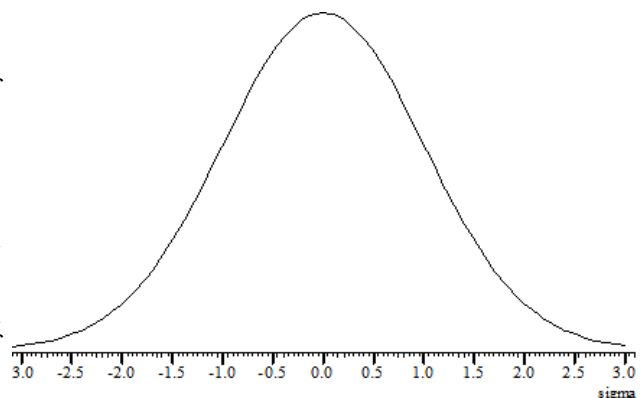
### Normal distribution

If you measure a data value in any real system, there is always some error in the measurement. Once you have made a (very) large number of measurements, you can form a curve showing the probability of getting any particular value. One would hope that this error distribution would show a large peak at the "correct" value of the measurement and the width of this distribution would show the spread of likely errors.

There is a particular error distribution that often occurs, called the Normal distribution. If a set of measurements is normally distributed, with a mean  $y_m$  and standard deviation  $\sigma$  (sigma), then the probability of measuring any particular value  $y$  is proportional to:

$$P(y) \propto \exp(-\frac{1}{2}(y-y_m)^2/\sigma^2)$$

It is for this distribution of errors that we have the well-known result that 68% of the values lie within one standard deviation of the mean, that 95% lie within two standard deviations and that 99.7% lie within three standard deviations. Of course, **if the error distribution is not normal, these results do not apply.**





## Chi-squared

The fitting routines given here define *best fit* as the values of  $\mathbf{a}_i$  (the coefficients) that minimise the chi-squared value defined as the sum over the measured data points of the square of the difference between the measured and predicted values divided by the variance of the data point:

If the sigma of each data point is unknown, then the fitting routines can be used to minimise the sum of the squares of the differences between the actual data values and the values predicted by the fitted function which produces the same result as a chi-squared fit would produce if the variance of the errors at all the data points was the same. This is commonly called least-squares fitting (meaning that the fit minimises the sum of squares of the errors between the fitted function and the data).

Chi-squared fitting is also a maximum likelihood fit if the errors in the data points are normally distributed. This means that as well as minimising the chi-squared value, the fit also selects the most probable set of coefficients that model your data. If your data measurement errors are not normally distributed you can still use this method, but the fit is not maximum likelihood.

If your errors are normally distributed and if you know the variance(s) of the data points, you can form good estimates of the variance of the fitted coefficients, and you can also test if the function you have fitted is likely to model the data.

If your errors are normally distributed but you do not know the variance of the errors at the data points, you can make an estimate of the variance of the errors (based on the assumption that the variance is the same for them all and that the model does fit the data), by fitting your model and calculating the variance from the errors between the best fit and the data. Having done this, you cannot then use this variance to test if the fit is likely to model the data.

## Residuals

Once your fit is completed, it is a good idea to look at the graph of the errors between your original data and the fitted data (the residuals or residual errors). If your errors are normally distributed and are independent, you would expect this graph to be more or less horizontal with no obvious trends. If this is not the case, you should consider if the correct model function has been selected, or if the fitting function has found the true minimum.

## Linear fit, Non-linear fit

A linear fit is one in which the theoretical model  $y = f(x, \mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \dots)$  can be expressed as  $y = \mathbf{a}_0 f_0(x) + \mathbf{a}_1 f_1(x) + \mathbf{a}_2 f_2(x) \dots$  for example  $y = \mathbf{a}_0 + \mathbf{a}_1 x + \mathbf{a}_2 x^2$ . Linear fits are relatively quick as they are done in one step. Usually, the only thing that can cause a problem is if the functions  $f_i(x)$  are not linearly independent. The methods we use can usually detect this problem, and can still give a useful result.

A non-linear fit means all other cases, for example,  $y = \mathbf{a}_0 \exp(-\mathbf{a}_1 x) + \mathbf{a}_2$ . We solve these types of problem by making an initial guess at the coefficients (and ideally providing a range of values that the result is known to lie in) and then improving this guess. This process repeats until some criterion is met. Each repeat is called an *iteration*, so we call this an iterative process.

## Covariance array

Several of the script fitting routines return a covariance array. If you have  $n$  coefficients, this array is of size  $n$  by  $n$  and is diagonally symmetric. If the errors in the original data points are normally distributed, the diagonal elements of this array are the variances in the values of the fitted coefficients. The remaining elements are the co-variances of pairs of the fitting parameters and can be used to estimate errors in derived values that depend on the product of two of the coefficients. If the errors are not normally distributed, the further away from normal the errors are, the less useful is the covariance array as a direct indication of the variances in the fitted coefficients.

For example, in the case of the linear fit:

$$y = \mathbf{a}_0 + \mathbf{a}_1x + \mathbf{a}_2x^2$$

you might collect your three coefficients in the array `coef[]`, and the covariance in the array `covar[][]`. In this case, the  $\mathbf{a}_0$  value is returned in `coef[0]` and its variance in `covar[0][0]`, the  $\mathbf{a}_1$  value is returned in `coef[1]` and its variance in `covar[1][1]`, and the  $\mathbf{a}_2$  value is returned in `coef[2]` and its variance in `covar[2][2]`.

Because the array is diagonally symmetric, `covar[i][j]` is equal to `covar[j][i]` and the off-diagonal elements are the expected variance in the product of pairs of the coefficients, so `covar[1][2]` is the variance of  $\mathbf{a}_1\mathbf{a}_2$ .

If you have not supplied the standard deviations of the errors in the data points, the covariance array is calculated on the assumption that all data points have a standard deviation of 1.0, and the covariance array is incorrectly scaled. In this case, if inspection of the residuals leads you to the conclusion that the function does indeed fit the data and that the errors are more or less the same for all values and not too far from normally distributed, then you can scale the covariance array to the correct values by multiplying all the elements of the array by the sum of squares of the errors between the data and the fitted values divided by the number of data points. If there are `nD%` data points and `nC%` coefficients and the sum of squares of the errors is `errSq`, then use `ArrMul(covar[][], errSq/(nD%-nC%))`; to rescale the covariance (see Numerical Recipes, Modelling of Data, Fitting Data to a Straight Line).

### What does the covariance mean?

Having fitted our data, we would like some idea of how the errors in the original data feed through to uncertainties in the values of the coefficients. The best way to do this is to obtain many sets of (x,y) data and fit our coefficient to each set. Then we can inspect the values of the coefficients and obtain a mean and standard deviation for each coefficient. However, this is very time consuming and may not be practical.

If the errors in the data are normally distributed (or not too far from this ideal case) and known, then the covariance array gives you some useful information. The square root of the covariance for a particular coefficient is the expected standard deviation in that value (given that the remaining coefficients remain fixed at optimum values). In script language terms, the standard deviation of `coef[i]` is `sqrt(covar[i][i])`.

In this case you would expect the coefficient to be within one standard deviation of the “correct” result 68% of the time, within 2 standard deviations 95% of the time and within 3 standard deviations 99.7% of the time.

## Testing the fit

If the errors in the original data are normally distributed and known (not calculated from the fit), and you know the chi-squared value for the fitted data, you can ask the question, “Given the known errors in the original data, how likely is it that you would get a value of chi-squared at least this large given that the data is correctly modelled by the fitting function plus normally distributed noise?” The answer is (at least in terms of the script language) that the probability is: `GammaQ((nData% - nCoef%)/2.0, chiSq/2.0)`; where `nData%` is the number of data points to be fitted, `nCoef%` is the number of coefficients that were fitted and `chiSq` is the chi-squared value for the fit. `GammaQ()` is the incomplete Gamma function.

If you want to follow this result up in a statistical textbook, you should look up *chi-squared distribution for n degrees of freedom*. In our case, we have `nData%-nCoef%` degrees of freedom.

If the fit is reasonable, you should expect a probability value between 0.1 and 1 (but be a bit suspicious if you always get values close to 1.0, as you may have overestimated the errors in the data). If the wrong function has been fitted or if the fit is poor you usually get a very small probability.

Intermediate values (0.0001 to 0.1) may indicate that the errors in the original data were actually larger than you thought, or they may indicate that the model does not explain all the variation in the data.

## Fitting routines

The `ChanFit...()` family of commands give a script the same capability as the Analysis menu Fit Data command.

|                             |                                                                                                                                              |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ChanFit()</code>      | This command associates a fit with a data channel in a time, result or XY view, performs the fit and returns basic fit information.          |
| <code>ChanFitCoef()</code>  | This gets and sets fit coefficients, coefficient limits and the hold flags. It is equivalent to the coefficient page of the Fit Data dialog. |
| <code>ChanFitShow()</code>  | This controls the display of the fitted data.                                                                                                |
| <code>ChanFitValue()</code> | This returns the value at a given x position of the fitted function.                                                                         |

The remaining fitting functions allow you to fit curves to data in arrays:

|                                                |                                                                                                                                                                                                                                                                                                                                                                                                                           |
|------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>FitPoly()</code>                         | This fits a polynomial of the form $y = \mathbf{a}_0 + \mathbf{a}_1x + \mathbf{a}_2x^2 + \mathbf{a}_3x^3 \dots$                                                                                                                                                                                                                                                                                                           |
| <code>FitLine()</code>                         | This fits a straight line to data in a time or result view.                                                                                                                                                                                                                                                                                                                                                               |
| <code>FitLinear()</code>                       | This fits $y = \mathbf{a}_0f_0(x) + \mathbf{a}_1f_1(x) + \mathbf{a}_2f_2(x) \dots$ where the $f_n(x)$ are user-defined functions of $x$ .                                                                                                                                                                                                                                                                                 |
| <code>FitExp()</code>                          | This fits multiple exponentials of the form $y = \mathbf{a}_0 \exp(-x/\mathbf{a}_1) + \mathbf{a}_2 \exp(-x/\mathbf{a}_3) \dots$                                                                                                                                                                                                                                                                                           |
| <code>FitGauss()</code>                        | This fits Gaussians of the form $y = \mathbf{a}_0 \exp(-1/2(x-\mathbf{a}_1)^2/\mathbf{a}_2^2) + \mathbf{a}_3 \exp(-1/2(x-\mathbf{a}_4)^2/\mathbf{a}_5^2) + \dots$                                                                                                                                                                                                                                                         |
| <code>FitSin()</code>                          | This fits multiple sinusoids of the form $y = \mathbf{a}_0 \sin(\mathbf{a}_1x + \mathbf{a}_2) + \mathbf{a}_3 \sin(\mathbf{a}_4x + \mathbf{a}_5) + \dots$                                                                                                                                                                                                                                                                  |
| <code>FitNLUser()</code>                       | This fits a user-defined function of the form $y = f(x, \mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2 \dots)$ to a set of data points where the $\mathbf{a}_i$ are constant coefficients to be determined. You must be able to calculate the differential of the function $f$ with respect to each of the $\mathbf{a}_i$ coefficients. This is the most general fitting routine, and also the slowest and most complex to use. |
| <code>GammaP()</code><br><code>GammaQ()</code> | Use these functions to calculate the error function $erf(x)$ , the cumulative Poisson probability function and the Chi-squared probability function. These are very useful when considering probabilities of fits associated with the normally distributed data.                                                                                                                                                          |
| <code>LnGamma()</code>                         | This calculates the natural logarithm of the Gamma function. It can be used to calculate the log of large factorials and is useful when working with probability distributions.                                                                                                                                                                                                                                           |
| <code>BetaI()</code>                           | The incomplete Beta Function comes up when you work with the Binomial Distribution, Student's T Distribution and the F-Distribution.                                                                                                                                                                                                                                                                                      |

## XY Views

XY views have a wide variety of uses, from displaying user-defined graphs to drawing pictures. XY views have the following features:

- One x axis and one y axis shared between all data channels in the XY view, so all the channels share the same space, so you can overdraw one channel with another.
- Up to 256 data channels allowed in the view.
- Each data channel is a list of (x,y) data points. The number of data points in a channel is limited only by available memory and drawing time. However, you can limit the number of data points on a channel, in which case new data points replace the oldest data points.
- The data points can be drawn with markers at each data point. The range of marker styles currently includes: dots, boxes, plus signs, crosses, circles (Windows NT only), triangles, diamonds, horizontal lines and vertical lines. The size of the markers can also be set, and they can be made invisible.
- The data points can be joined with solid, dotted or dashed lines, and the line thickness can be varied. You

can also choose to join the last point in a channel to the first point to make a loop.

- You can sort the order of the data points in a channel by x, by y or by order of insertion in the channel. This is only important if the data points are joined.
- The colour of the lines and markers can be chosen. If no colour is set, the same colour as for a waveform channel in a time view is set.

There are several example scripts included with Spike2 that illustrate some of the uses of XY views. You will find them in the `scripts` folder:

|                       |                                                                                                                   |
|-----------------------|-------------------------------------------------------------------------------------------------------------------|
| <code>FFTWater</code> | This draw a “waterfall” type display showing the variation in the power spectrum of a waveform channel with time. |
| <code>DE</code>       | A script that uses a genetic algorithm to fit a double exponential.                                               |
| <code>HRaster</code>  | This shows a raster display and a PSTH of event data with the raster drawn above the PSTH.                        |
| <code>Clock</code>    | An analogue clock for your Toolbar idle routine.                                                                  |
| <code>PingPong</code> | A very silly example indeed.                                                                                      |

## Creating an XY view

Although you can create an XY view from the File New menu command, the usual way to generate XY views is with the Analysis menu Measurements command or with the script language. The script language can generate an XY view for general use with the `FileNew()` command and as the target for measurements with a `Process()` command with the `MeasureToXY()` command. See the documentation for `MeasureToXY()` for an example.

The following assumes that you have some familiarity with the script language. An XY view always has at least one data channel, so when you create a view, you also create a channel. The following script code shows you how to make an XY view:

```
var xy%; 'handle for the XY view
xy% := FileNew(12,1); 'type 12=XY, 1=make visible now
```

If you want to add additional channels you can do this using the `XYSetChan()` command. You can also use this command to set a channel to a particular state. The following sets channel 1 (the first channel) to show data points joined by lines with no limit on the number of data points, drawn in the standard colour:

```
XYSetChan(1, 0, 0, 1); 'chan 1, no size limit, no sort, joined
XYDrawMode(1, 2, 0); 'set a marker size of 0 (invisible)
```

To add data points to a channel you use the `XYAddData()` command. You can add single points, or pass an array of x and y co-ordinates. The following code adds three points to draw a triangle:

```
XYAddData(1, 0, 0); 'add a point to channel 1 at (0, 0)
XYAddData(1, 1, 0); 'add a point at (1,0)
XYAddData(1, 0.5, 1); 'add a point at (0.5, 1)
```

You will notice that the result of this draws only two sides of a triangle. We could complete the figure by adding an additional data point at (0,0), but it is just as easy to change the line joining mode to "Looped", and the figure is completed for you:

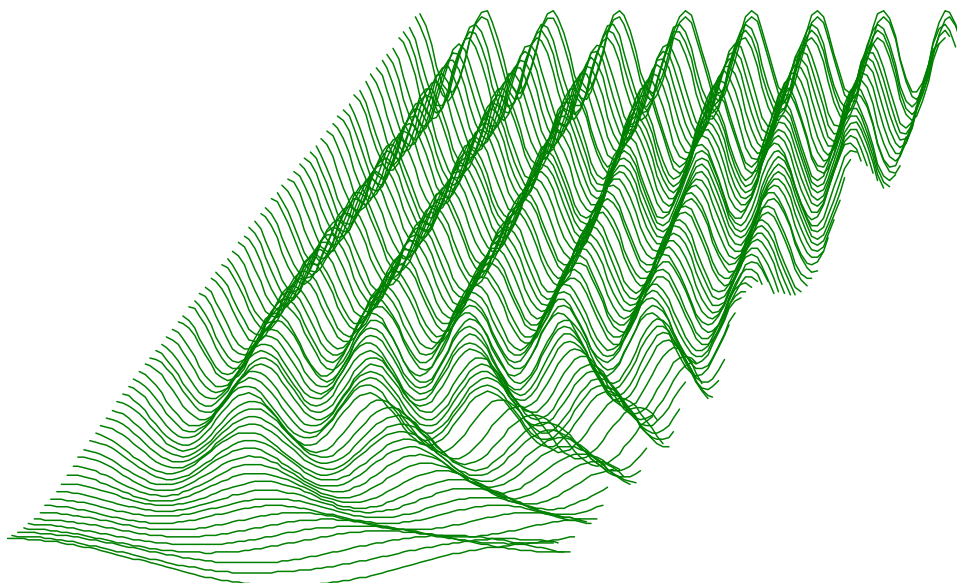
```
XYJoin(1,2); 'set looped mode
```

### See also:

`MeasureToXY()`, `FileNew()`

## Overdrawing data

You can use the XY view to overdraw data. For example, suppose we have a waveform channel and another channel of trigger markers and we want to superimpose the first 100 data points after each trigger. The following example does this. To make it a little easier to see, we have displaced each triggered data sweep slightly to the right and up. We have also turned off the x and y axes.



The code required to produce this image is quite short and can be easily adapted to display other types of data. This example expects to run with the `demo.smr` data file in the `Spike2\Data` folder created when you installed the program. To run with other files you would need to adjust the channel numbers.

The first line declares variables to remember the time and XY window handles (identifiers) and a variable to hold the number of points read. The second line checks that the current view is a time view and gives up if it is not. The third line remembers the time view handle.

```
var tv%, wh%, np%; 'time view, XY view, points read
if ViewKind() then Message("Not a time view");Halt endif;
tv% := View(); 'save view handle
```

The next three lines declare an array to hold 100 data points, a variable to hold the trigger time (and we set it to -1 so we see the first event, even if it is at time 0), and we call the `MakeWindow%` function (see below) that creates the XY window for us. We set the first argument to the time interval between data points in our time view so that the x spacing of the data points is correct. The second and third arguments to this function set the distance each “slice” of data is moved to the right and up.

```
var wave[100]; 'space for 100 waveform data points
var t := -1; 'start time for trigger search
wh% := MakeWindow%(BinSize(1), 0.01, 0.5); 'start waterfall view
```

To generate the data we run round a loop (`repeat...until (t<0) or (t>20);`) that finds the time of the next trigger on channel 2, reads in 100 data points into the array `wave[]` and then as long as the data read correctly, adds the data into the XY window.

```
repeat
 View(tv%); 'Make time view current
 t := NextTime(2, t); 'Get next trigger time from channel 2
 np% := ChanData(1, wave[], t, MaxTime()); 'get channel 1 data
 if (np% > 0) then AddSlice(wave[:np%]) endif; 'add to picture
until (t < 0) or (t > 20); 'until end or we reach 20 seconds
```

The final task is to make the XY window visible and halt.

```
View(wh%).WindowVisible(1); 'make XY window visible
halt;
```

The remaining code can be copied from the `FFTWater.s2s` script. The variables with names starting `WF` are used to remember the initial waterfall settings. There are two functions in this code. `MakeWindow%` prepares for the waterfall display by creating the XY window and storing the information needed to position the channels. `AddSlice()` takes an array of data points and adds it to the display as an additional channel.

```
'===== Waterfall display code =====
var WfXInc,WfYInc,WfBinSz,WfSlices%,WfVh%;
'
'xBinSz Width of each bin (sample interval)
'xInc Add to each slice x co-ords to give waterfall effect
'yInc Add to each slice y co-ords to give waterfall effect
Func MakeWindow%(xBinSz, xInc, yInc)
```

```

WFSlices% := 0; 'no slices yet
WFxInc := xInc; 'save x increment per slice
WFyInc := yInc; 'save y increment per slice
WFbinSz:= xBinSz; 'save data point separation
WFvh% := FileNew(12); 'create a new XY window (hidden)
return WFvh%; 'return the XY window handle
end;

Func AddSlice(y[]) 'Add data to the waterfall
View(WFvh%); 'select the waterfall view
var ch%=1; 'true if this is the first channel
if WFSlices% = 0 then 'if first channel no need to create
 XYSetChan(1,-Len(y[]),0,1); 'set original channel
else
 ch% := XYSetChan(0, -Len(y[]), 0, 1); 'create new channel
 if (ch% <= 0) then return ch% endif; 'No more channels
endif;
WFSlices% := ch%; 'number of slices
XYDrawMode(ch%,2,0); 'Hide the markers (set size of 0)
var x[Len(y[])]; 'space for x values, same size as y[]
ArrConst(x[], WFbinSz); 'generate x axis values, set the same
x[0]:=(ch%-1)*WFxInc; 'set x offset as first value
ArrIntgl(x[]); 'form the x positions
ArrAdd(y[],(ch%-1)*WFyInc); 'add the y offset to the y array
XYAddData(ch%, x[], y[]); 'add the (x,y) data points
return 1; 'return >0 means all OK
end;

```

## Channel offsets

At Spike2 version 7.09 we added the ability to shift a channel in an XY view with the `XYOffset()` command. This command moves the XY view origin to wherever you specify when the channel is drawn. You set the offset in axis units. That is, if the y axis is in Logarithmic mode and the x axis is linear, and you want the origin to be at the point where (x,y) would appear, you must set the offset to (x, Log(y)). Sadly, this means that the offsets must be recalculated if the axis mode is changed.

The `XYInRect()`, `XYInCircle()` and `XYInChan()` functions can be used when channels are offset and tell you which visible points lie inside the rectangle, circle or channel. If you do not want to take offsets into account you must set the channel offset to (0,0). The `XYRange()` function now has an extra argument to specify if the offset is to be considered or ignored.

The previous example of a waterfall plot can be done using channel offsets by changing the `AddSlice()` function:

```

Func AddSlice(y[]) 'Add data to the waterfall
View(WFvh%); 'select the waterfall view
var ch%=1; 'true if this is the first channel
if WFSlices% = 0 then 'if first channel no need to create
 XYSetChan(1,-Len(y[]),0,1); 'set original channel
else
 ch% := XYSetChan(0, -Len(y[]), 0, 1); 'create new channel
 if (ch% <= 0) then return ch% endif; 'No more channels
endif;
WFSlices% := ch%; 'number of slices
XYDrawMode(ch%,2,0); 'Hide the markers (set size of 0)
var x[Len(y[])]; 'space for x values, same size as y[]
ArrConst(x[1:], WFbinSz); '[0] left as 0, rest set the same
ArrIntgl(x[]); 'form the x positions
XYOffset(ch%, (ch%-1)*WFxInc, (ch%-1)*WFyInc);
XYAddData(ch%, x[], y[]); 'add the (x,y) data points
return 1; 'return >0 means all OK
end;

```

In this version, the x co-ordinates are the same for each histogram (and could be calculated once, saving a little time). The y co-ordinates can be left as read; they do not need offsetting. If you wanted to dynamically change the angle of the waterfall, you could remove the `XYOffset()` call from the `AddSlice()` routine and instead have a separate routine:

```
Proc SetAngle(xInc, yInc, nCh%)
```

```
var ch%;
for ch% := 1 to nCh% do
 XYOffset(ch%, (ch%-1)*xInc, (ch%-1)*yInc);
next;
end;
```

The way this example is written, the slices are added to the front first, working backwards. If you wanted to draw the data as a histogram (another new XY view feature in version 7.09, see `XYJoin()`), you would want to draw the data from the back to the front as channels are drawn in order and you would want the last channel to be at the front so that the filled histograms overwrite each other in a sensible way. If you did this, then in `SetAngle()`:

```
XYOffset(ch%, (nCh%-ch%)*xInc, (nCh%-ch%1)*yInc);
```

so that the last channel (at the front) has no offset and the first channel (at the back) has the largest.



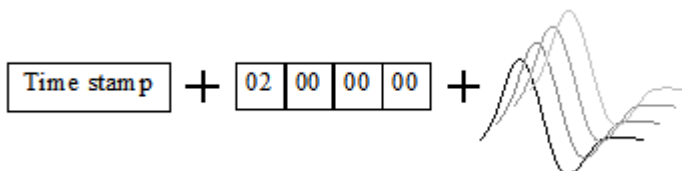


# **16: Spike sorting**

# Spike sorting

## Introduction

Spike2 stores spike shapes as WaveMark channels. Each spike shape has a time stamp, 4 identifying codes and 1, 2 or 4 traces of up to 1000 waveform data points that define the spike (on-line you can use up to 126 points). The codes store the spike classification, obtained by matching the spike shape against shape templates or by using cluster analysis; normally only the first code is used. The time stamp is the time of the first saved data point.



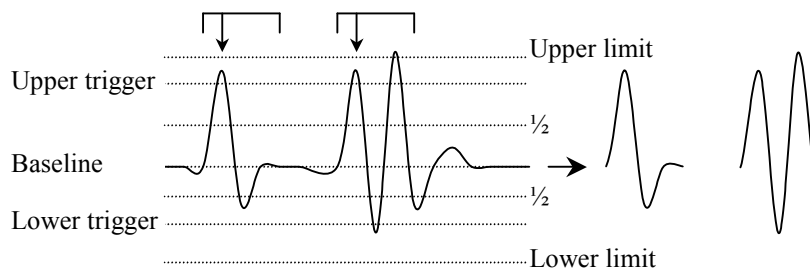
On-line, spikes can be sorted in real-time with up to 8 spike classes per channel and an optional digital output as each spike is classified. Spike2 captures up to 32 channels of spikes with a Power1401 and up to 16 channels with a Micro1401. Off-line Spike2 can extract spikes from a waveform channel and resort and edit spikes extracted on-line or off-line; you can divide spikes into almost any number of classes. Off-line spike sorting can be script-controlled.

The first part of this section is organised as reference information. This is followed by *Getting started with spike shapes and templates* which is more discursive. If you are a new user you may prefer to start there and come back to the reference material later.

## Spike detection

Spikes are detected by the input signal crossing a trigger level. There are two trigger levels, one for positive-going and one for negative-going spikes. Captured spikes are aligned on the first positive or negative peak. To avoid problems due to baseline drift, the data capture routines pass the data through a high pass filter. The time constant of this filter can be adjusted to suit the data. You can define the number of data points captured before and after the peak. The spike detection algorithm is as follows:

1. Wait for the signal to lie within half the trigger levels. When it does, go to step 2.
2. Wait for the signal to cross either trigger level. If it crosses the upper trigger level go to step 3. If it crosses the lower level, go to step 4.
3. Track the positive peak signal value. If the signal falls below the peak and there are sufficient post-peak points to define a spike, go to 5. If the signal falls below half the upper trigger level, we ignore further peaks (as for the second spike in the diagram).
4. Track the negative peak signal value. If the signal rises above the peak, see if we have sufficient post-peak points to define the spike. If so, go to 5. If the signal rises above half the lower trigger level, we ignore further peaks.
5. Save the waveform and first data point time and go to step 1 for the next spike.



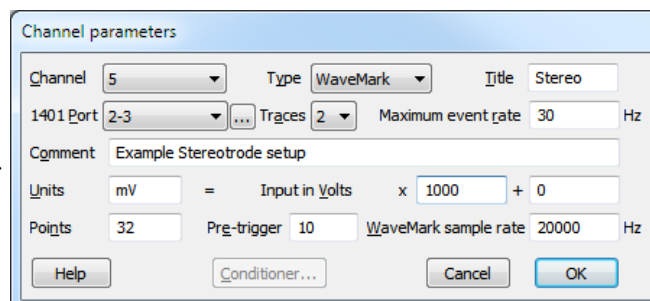
With multiple traces, each trace is tracked separately and the first trace to trigger saves the data for all traces.

When working offline and with Power1401s and the Micro1401-3, you can choose to set two additional limit

levels outside the trigger levels. Any detected spikes with a peak amplitude outside these limits are rejected. In this case the trigger and associated limit act as an amplitude window on the data. The limit applies to the initial peak amplitude, so the second spike in the diagram is acceptable, even though some of the data points lie outside the limits.

## WaveMark channel settings

To sample spike shapes you must create a WaveMark channel in the Sampling configuration dialog. The Title, Port, Channel comment, Units and Scaling fields are the same as for Waveform data. The Maximum event rate field is the total number of spikes (of all classes) that you expect per second on this channel.



The Traces field sets the number of waveforms that are combined in this channel. This should be set to 1 for a single electrode, 2 for a stereotrode and 4 for a tetrode. The 1401 Port sets the input channel number of the first trace. The remaining traces are taken from sequential channel numbers. For example, if the 1401 Port field was set to 7 and Traces was 4, the 1401 input channel numbers 7, 8, 9 and 10 would be used for the traces.

The bottom line of the dialog box sets the number of points to save per trace per spike in the range 10 to 126 (Points) and the number of points to display before the peak (Pre-trigger). You can alter the number of points before the peak and the total number of points in the template setup section, so it is not essential to get this exactly right.

The WaveMark sample rate field sets the ideal sample rate for all WaveMark channels. A sampling rate of 20 kHz per channel is often used so that spikes (of typical duration 1 to 2 milliseconds) can be usefully discriminated. The actual sampling rate depends on the number of waveform and WaveMark channels. See the Sampling configuration Resolution tab for more information on setting waveform and WaveMark sample rates.

## Template matched signal

You can use digital output bits 0-7 (on the rear panel of the Micro1401 and Power1401) to signal that a spike shape has matched a template. There are two output formats (set in the on-line Template Setup). In One bit mode, a single bit pulses from low to high, then back to low for each of up to 8 templates. Bit 0 is for the first template, bit 1 for the second, and so on up to bit 7 for the eighth template. One bit mode is not useful when you have more than one channel of spikes (WaveMark data) as you cannot tell which channel produced the output.

In Coded mode, pin 23 of the digital output port pulses for each matched template, and the digital output bits 3-7 hold the ADC port number (0 to 31) and bits 0-2 hold the template (0-7 for the 8 possible templates). The data is valid on both edges of the pulse. The pulse is short, less than 1 microsecond for the Power1401, more than 1 microsecond for other 1401s. If you have more than 32 ADC channels, the bottom 5 bits of the port number are presented on bits 3-7.

### Digital output connections

|            |   |    |   |    |   |    |   |    |     |        |
|------------|---|----|---|----|---|----|---|----|-----|--------|
| Data bit   | 7 | 6  | 5 | 4  | 3 | 2  | 1 | 0  | Gnd | Strobe |
| Output pin | 5 | 18 | 6 | 19 | 7 | 20 | 8 | 21 | 13  | 23     |

Coded mode uses all 8 bits, One bit mode uses one bit per template. The Micro1401 and Power1401 have independent input and output bits 0-7, however you should use the output sequencer DIGLOW command with caution as it controls the same output bits.

### Timing of the digital output

Measurements with templates of 20 points show that the vast majority of spikes are matched within a

millisecond (often within a few tens of microseconds) of the end of the spike (less than this with a Power1401). However you cannot rely on it. Even with a fast host, longer delays may occur occasionally, especially when data is written to disk at high sampling rates. At very high data rates, if the 1401 cannot keep up, it may stop matching spikes to templates until the data rate drops.

You can test the delays by feeding the output back into an event channel and correlating the Spike channel with the event channel. If you intend to use these outputs for a timing critical use it is important that you measure the typical time delays for your configuration.

### Power1401-3 One bit pulse width

Beware that the `One bit` output from a Power1401-3 is very short (0.1 us). If you want to record this signal with the Power1401-3 by connecting it to an event input (for example to verify the digital output timings) you must use the rear-panel digital inputs, not the more convenient front panel Event inputs. This is because the front panel inputs have deglitching circuits that remove pulses shorter than around 0.25 us. The rear panel inputs are not deglitched in this way so can be used to time the One bit mode output.

## Spike shape sorting dialogs

There are three similar dialogs that are used to sort spikes by shape:

### On-line template setup

This dialog opens automatically when you open a data file for sampling with one or more WaveMark channels in the sampling configuration. You can control if this dialog appears with the `FileNew() mode%` parameter when you create a file for sampling using the script language.

### Off-line template formation

This dialog is opened from a time window by right-clicking on a waveform or WaveMark channel and selecting the `New WaveMark` command or by using the Analysis menu `New WaveMark` or `New N-Trode` commands. From the dialog you can create a new WaveMark channel from one or more source channels.

### Edit WaveMark (on-line/off-line)

This dialog can be used on-line to monitor spikes, adjust the trigger levels and control template formation. Off-line it can be used to resort WaveMark channels and to create new WaveMark channels. This dialog is also the starting point for Spike Clustering.

## Menus in Spike Shape sorting dialogs

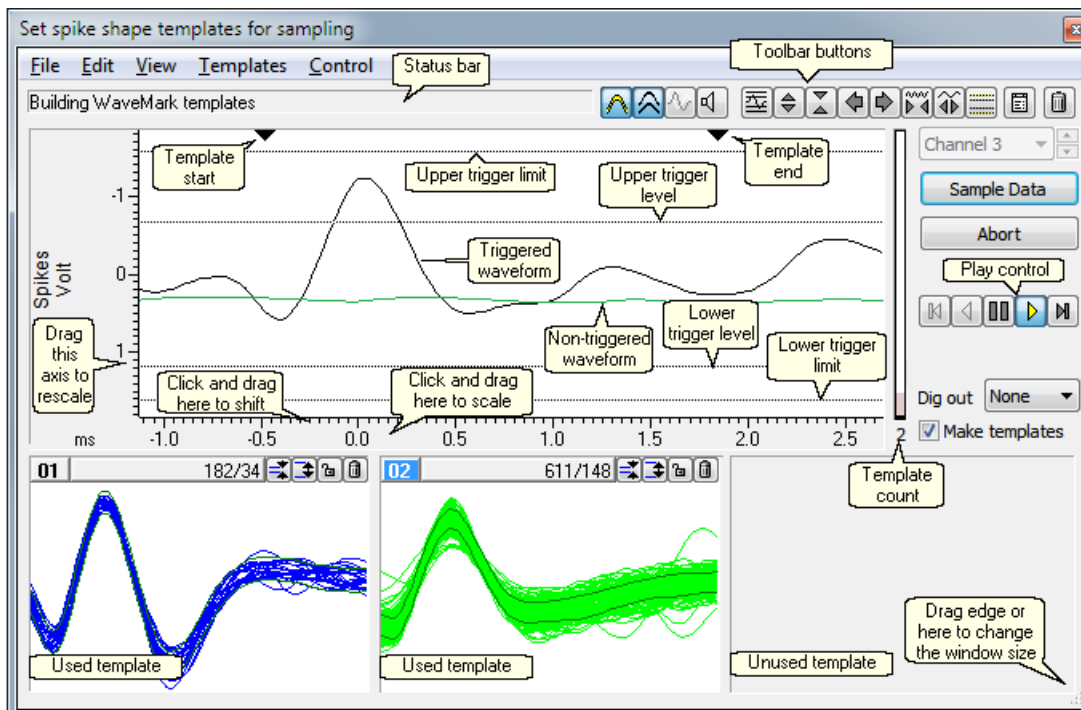
The three Spike shape dialogs (Setup Sampling, New WaveMark and Edit WaveMark) have a very large number of controls and options, most of which are available from buttons and short-cut keys. The menu system in the dialog provides access to commands that are not available on buttons, and also duplicates many of the other controls, for completeness.

| Menu        | Command             | Setup | New | Edit | Comment                                |
|-------------|---------------------|-------|-----|------|----------------------------------------|
| <b>File</b> | Load and Save       | Yes   | Yes | Yes  | Load and Save templates                |
|             | Sample Data         | Yes   |     |      | Accept current dialog settings, sample |
|             | Abort Sampling      | Yes   |     |      | Close dialog and abandon sampling      |
|             | Print...            |       | Yes | Yes  | Print current set of templates         |
|             | New Channel         |       | Yes |      | Create a new channel from the input    |
|             | Close               |       | Yes | Yes  | Close the dialog                       |
| <b>Edit</b> | Clear All Templates | Yes   |     |      | Delete all templates                   |

|                     |                         |     |     |                                        |                                            |
|---------------------|-------------------------|-----|-----|----------------------------------------|--------------------------------------------|
| <b>View</b>         | Copy...                 | Yes | Yes | Yes                                    | Copy current templates as image or text    |
|                     | Overdraw                | Yes | Yes | Yes                                    | Overdraw templates in the template area    |
|                     | Show Template Limits    | Yes | Yes | Yes                                    | Show/hide template limits in template area |
|                     | Show Non-Matching       | Yes | Yes | Yes                                    | Show non-matched spike on all templates    |
|                     | Overlay Traces          | Yes | Yes | Yes                                    | Overdraw multiple traces or side by side   |
|                     | Optimise                | Yes | Yes | Yes                                    | Optimise the y axis of the display         |
|                     | Show All Y Range        | Yes | Yes | Yes                                    | Show full input range of the signal        |
|                     | Scale Up/Down           | Yes | Yes | Yes                                    | Make templates bigger/smaller              |
|                     | Pan Left/Right          | Yes | Yes |                                        | Change the number of pre-trigger points    |
| Enlarge/Reduce View | Yes                     | Yes |     | Change number of displayed data points |                                            |
| <b>Templates</b>    | Make Templates          | Yes | Yes | Yes                                    | Create new templates from spike shapes     |
|                     | Sound                   | Yes | Yes | Yes                                    | Play a tone when a spike is displayed      |
|                     | Renumber                | Yes | Yes | Yes                                    | Renumber the templates                     |
|                     | Order by Code           | Yes | Yes | Yes                                    | Sort displayed templates into code order   |
|                     | Initialize Counter      | Yes | Yes | Yes                                    | Reset the matched template counters to 0   |
|                     | Template Size           | Yes | Yes | Yes                                    | Small, medium or large template size       |
|                     | Parameters              | Yes | Yes | Yes                                    | Open the Template Parameters dialog        |
|                     | Form Templates          |     |     | Yes                                    | Form templates by shape or by code         |
| <b>Control</b>      | Step Back/Forwards      | Yes | Yes | Yes                                    | One spike forwards (back where allowed)    |
|                     | Run Back/Forwards       | Yes | Yes | Yes                                    | Run forwards (backwards where allowed)     |
|                     | Pause                   | Yes | Yes | Yes                                    | Stop moving through spikes                 |
|                     | Digital Output          | Yes |     |                                        | Online digital output on template match    |
|                     | High level cursors      |     | Yes | Yes                                    | Use outer cursors to exclude large spikes  |
|                     | Circular Replay         |     | Yes | Yes                                    | Play data in a loop; don't stop at the end |
|                     | Set Time Range...       |     | Yes | Yes                                    | Set time range of data to process          |
|                     | Jump to Start           |     | Yes | Yes                                    | Jump to the start of the time range        |
|                     | Track Cursor 0          |     | Yes | Yes                                    | Time view scrolls so Cursor 0 is centred   |
|                     | Always At End           |     |     | Yes                                    | Online only, display new spikes            |
| <b>Analyse</b>      | Principal Components    |     |     | Yes                                    | Open Principal Components dialog           |
|                     | Cluster on Measurements |     |     | Yes                                    | Open Cluster on Measurements dialog        |
|                     | Cluster on Correlations |     |     | Yes                                    | Open Cluster on Correlations dialog        |
|                     | Cluster on Errors       |     |     | Yes                                    | Open Cluster on Errors analysis dialog     |
|                     | Collision Analysis Mode |     |     | Yes                                    | Attempt to separate spike collision mode   |
|                     | Duplicate               |     |     | Yes                                    | Make duplicate channel for each template   |
|                     | Set Codes               |     |     | Yes                                    | Opens the Set Marker Codes dialog          |
|                     | Marker Filter           | Yes | Yes | Yes                                    | Display marker filter dialog               |

## On-line template setup

When you create a New file with WaveMark channels, Spike2 opens the template setup window. The window has its own menu and contains four main regions: a status and tool bar at the top, a data display area with the trigger levels, a control area on the right and a region with up to 20 templates at the bottom.



The display area shows data from the channel set by the channel control. Waveforms that meet the trigger criteria are drawn in the WaveMark 00 colour; un-triggered waveforms use the time view waveform colour.

The horizontal dotted lines are the trigger (and limit) cursors; click and drag them to change the levels. You can display 2 or 4 trigger levels. With 4, the outer ones act as limits; to trigger a wave must cross the trigger and not the limit. To display level cursors in one direction only, right-click an unwanted cursor and select **Move away**. To restore cursors that are off the visible area right click in the display area and select **Fetch cursors**.

The two black triangles at the top of this area mark the start and end of the region used for templates and principal component analysis.

The vertical bar on the right of the data display shows the number of templates that the program is considering (provisional templates) in grey, with black representing confirmed templates. If the number of confirmed and provisional templates gets large, the provisional template colour becomes red. If this occurs, you may need to adjust the values in the Template Settings dialog to be more tolerant of variation in the spikes.

You can change the window size by dragging any window edge. Double click the title bar of the window to maximise (or minimise if already maximised) the window. The window size can be changed from very small (minimised), with no templates, to large (maximised) with all 20 possible templates (though only the first 8 are used on-line). The templates can be small, medium or large. Double click on one of the template rectangles to change the template size or use the **View** menu.

## Channel control

When there is more than one channel holding WaveMark data, you can change the channel either by dropping down the list of channels and selecting one, or by using the small spin box on the right of the channel control to move to the next channel in the list. Each channel has its own set of templates and template parameters.

## Sample data

Once you have adjusted the trigger levels and set the templates for each channel, click this button to close the window and sample data.

## Abort

This button closes the window and does not sample data.

## Dig out

During sampling, the 1401 can flag template matches using bits 0-7 of the digital port. You can select between **None** (no output), **One bit** (single pulsed digital bit per template) or **Coded** (data channel and template coded in the 8 data bits).

## Make templates

With this box checked, detected spikes are offered to the template matching system and will create and modify templates. With this box unchecked, spikes are compared against templates but the templates do not change. When you open this window, this box is checked if there are no existing templates, otherwise it is clear. This command is also available in the **Templates** menu with a keyboard short-cut of `Ctrl+M`.

## Play controls



These five buttons control data replay. The centre button pauses replay and the buttons either side of it play the data forwards and backwards. The outer buttons step one spike forwards and backwards. You can only play or step backwards in the **Edit WaveMark** command. There are keyboard shortcuts for these controls. `B` steps backwards one spike, `M` steps forwards one spike, `P` runs forwards, `Q` runs backwards and `V` pauses output. You can also access the **Play** commands from the **Control** menu.

## Renumber...

This command is available from the **Templates** menu with a keyboard short-cut of `Ctrl+R`. It prompts you for a starting marker code then renumbers the templates with consecutive codes, starting with the code you set. The templates remain in the original positions and code order. If two or more templates share a marker code they will still share a code after renumbering. For example, templates coded 01, 04, 04, 07 and 03 renumbered to start at 01 would end up with codes 01, 03, 03, 04 and 02.

## Order by code

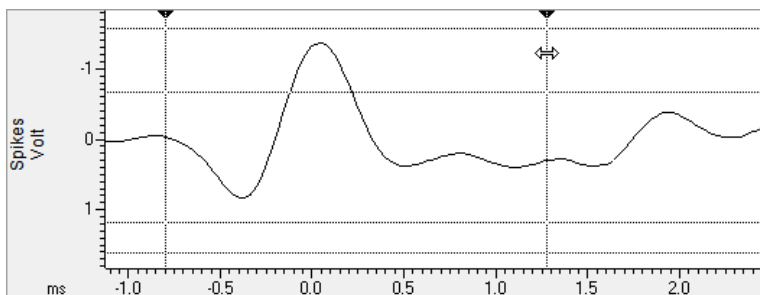
This command is available from the **Templates** menu with a keyboard short-cut of `Ctrl+B`. It rearranges the templates so that they are in ascending marker code order. For example, templates with marker codes 01, 04, 04, 07 and 03 would be sorted into the order 01, 03, 04, 04 and 07. The sorting order is left to right and top to bottom.

## Load and Save...

This command is available from the **File** menu with a keyboard short-cut of `Ctrl+S`. It opens a new dialog in which you can load templates generated in previous spike sorting sessions and save your templates to resource files.


## Selecting the area for a template

The two black triangles at the top of the large display area set the waveform region to use for template formation and for principal component analysis. To adjust this region, click and drag the triangles with the mouse. Vertical cursors appear as guides for the new template position. Old templates are deleted if the size of the template has changed. The vertical cursor positions and the number of data points that will be used in the template appear in the status bar when you activate the vertical cursors.



The best information about the spike class is usually around the peak; the start and end of the spike holds

mainly baseline noise. If you use the entire spike shape for template matching, you will be comparing baseline noise for some of the record, which is a waste of time and reduces the sensitivity of the template matching.

If your spike fills a small proportion of the spike trace (as in the example above), you should reduce the number of points for the WaveMark channel with the  button or by dragging the x axis. This has several advantages: reduced disk space for storage, faster processing, quicker drawing of data and fewer records with a second spike overlapping the end.

If you have too few points around the interesting region of your spike you will need to increase the sampling rate. Remember that the template routines interpolate between data points, so you only require enough points to define the spike shape.

You will get the best spike discrimination when the area selected shows the greatest variation between different classes of spikes. For example, if all your spike classes look very similar at the end, you will improve discrimination by excluding the end of the data.

## Horizontal cursors

There are horizontal cursors in the Spike shape dialogs except in the case of the off-line Edit WaveMark dialog. There is a choice of 2 cursors (setting upper and lower trigger levels) or 4 cursors (setting amplitude limits and trigger levels) controlled by a toolbar button. You can drag these cursors; when you drag, the cursor positions appear in the status bar at the top of the dialog.

### Linked cursors

You can link the horizontal cursors to horizontal cursors 1 to 4 in the associated time view. When linked, horizontal cursor 1 is the lower trigger level, 2 is the upper trigger, 3 is the lower limit and 4 is the upper limit. The option is checked when linked. When you set this option, cursors corresponding to the visible cursors in the spike shape dialog are moved to (or created in) the same positions in the source time view channel.

When cursors are linked, each time you release a cursor after dragging with the mouse, the linked cursor will also move. As you can drag time view cursors to any position or channel, only moves to sensible positions on the same channel will have any effect on the spike shape window.

Any horizontal cursors in the time view that did not exist before the link are considered to be owned by the spike shape dialog and will be deleted if the spike shape dialog does not need them. If you change channel in the spike shape dialog while linked, all owned cursors are deleted from the time view, then the dialog behaves as if you had requested a link command for the new channel.

### Caveats

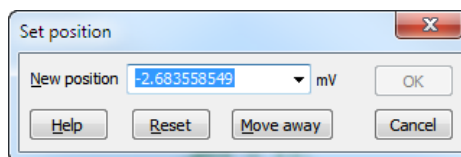
For this feature to be useful, the source channel must have the same DC offset as the spikes you see in the Spike shape window (usually 0). This will be the case for WaveMark source channels (for example when sampling data), but need not be the case with Waveform or RealWave data. Further, the mean level of the spike data must be the same as the DC offset (usually 0). If the source data does not have the same DC offset, the cursors will still work and the values will match, but the source data will be vertically shifted. You can often work around this by applying a DC remove process to the source channel. This is an experimental feature and we do not store the status of the link when the spike shape dialog closed.

## Context menu for the cursor area

There is a context menu (right-click) for the data display area to control the horizontal cursors. The first three context menu items only appear if you click on a cursor:

### Set cursor position...

This opens the Set position dialog. Click **Reset** to restore the original position. **Move away** closes the dialog and moves the cursor to a position where it will have no effect on spike capture. The drop down list contains the original cursor position and also allows you to select horizontal cursor positions in any associated time view(s). **OK** is enabled if the new position is legal and differs from the original.





**Copy position 2.345 mV**

Select this option to copy the current cursor position and units to the clipboard as text.

**Move away**

Move the cursor to a position where it has no effect on spike triggering. If you have enabled the amplitude limit cursors and you use the Move away command on a trigger level, both the trigger and associated amplitude cursor will move away. If you move away a linked cursor the time view linked cursor will be deleted if it was created by linking, otherwise it will move out of the way.

**Fetch cursors**

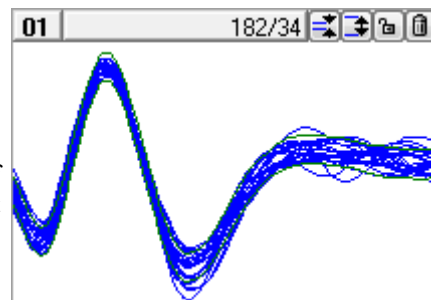
This option appears if you right-click in the WaveMark display and a horizontal cursor is beyond the displayed y axis range. This command moves the cursor back into a visible range. If there are multiple traces, only cursors in the trace on which you click are restored. If you are linked to a time view, fetched cursors will become visible in the time view.

**Link Time view cursors**

This option links and unlinks the visible horizontal cursors in the spike shape dialog to horizontal cursors 1 to 4 in the time view channel that sources the data. This option is disabled if the Cursor menu is disabled during script operation.

## The template area

Templates appear at the bottom of the window. If this area is not visible, drag the bottom edge of the dialog downward until the template area appears. Templates can be displayed in three sizes: small, large or huge; double click a template to cycle round the three sizes. The image shows a medium sized template; small templates have less information at the top. The template code is at the top left of the template area. When a template first appears, it has the lowest unused code. Double click the template code to open a dialog where you can change the code. You are allowed to have multiple templates with the same code.

**Event count**

The count of spikes that matched the template is in the centre. In large and huge mode, the display also shows the number of spikes added to the template as: matched/added. You can set the count of matched events to 0 with the Template menu Initialise Counters command; the short-cut key is `Ctrl+I`.

The display area holds the last spike that matched the template, the template itself and optionally all non-matching spikes. You control the drawing style of the matching and non-matching spikes and the template with buttons at the left-hand end of the toolbar. These buttons can appear at the top of each template area:

**Change width** 

These buttons are hidden in small template windows. They decrease and increase the width of the template.

**Lock** 

This button locks or unlocks a template. A locked template does not change when new spikes match it. An unlocked template adapts as each new spike is added. Templates can lock automatically if Auto Fix mode is set in the template settings dialog. When building templates, Spike2 cannot merge a provisional template with a locked template. Instead, Spike2 creates a new template with the same code as the one it would have merged with.

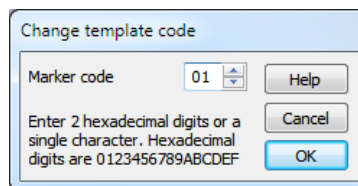
**Clear** 

This button erases the template. To erase all templates, click the bin icon in the toolbar.

## Edit template code


You can open this dialog by double-clicking a template code at the top-left of a template or one of the four template codes on the right-hand side of the spike sorting dialog.

Double-digit codes are read as hexadecimal marker codes, single characters are read as printing characters. If you have a spike that changes shape too much for one template to represent it, let it generate several templates and then edit the codes to be the same. You can use code 00 to mark spikes or artefacts you are not interested in.



## Merging templates

You may decide that two templates are so similar that you would like to merge them into one. To do this, drag one of the templates over the one that you wish to merge it with and release the mouse button. The dragged template vanishes and the template over which you released it changes to take account of the new data.

 The mouse pointer is an open hand when it is over a spike or template that you can drag. To drag, click and hold the left mouse button; the pointer changes to a closed hand. When you drag over a drop zone, the closed hand has a plus sign on the back. You can compare templates by dragging, but not releasing the mouse. If you decide not to merge, make sure the mouse pointer is not the closed hand with the plus sign when you release the mouse button.

## Manual template creation

When paused, you can use drag and drop to create new templates and to classify spikes. Click the mouse in the data display area and drag the current spike to an existing template to set the marker code, or to an unused template to create a new template.

You can also use the keyboard to do the same thing: The keys 1 to 9 will match the current spike to the first template with the code 01 to 09, or if there is no template with a matching code, a new template will be created with that code. The 0 key can be used when editing spikes to give the current spike the code 00.

## Toolbar controls

The toolbar buttons can be clicked with the mouse to control the template forming process. The leftmost group are state buttons; click them to set the state, click them again to remove the state. The remaining buttons act immediately. These commands are also available in the dialog menus and most have short-cut keys. The buttons are:



### Overdraw

Normally, the templates display the last spike added. You can choose to show all spikes added to templates by overdrawing. This omits the step of erasing the previous spike before drawing a new one. The short-cut key is `Ctrl+O`.



### Template Limits (shape)

You can choose to display the mean template, or the upped and lower limits of the template. The template limits are useful when you display the last spike and need to see how well it fits within the template. The mean template can be useful when you are comparing template shapes. The short-cut key is `Ctrl+L`.



### Show non-matching spikes

In some circumstances you need to see every spike displayed over every template. With this button down, all non-matching spikes are drawn on top of the template in the "Not saving to disk" colour. The short-cut key is

Ctrl+N.



### Sound on/off

It can be useful to have an audible indication of each spike as it is added to a template. If you have a sound card, turn this on to hear a tone for each spike. A different tone is played for each template code; the tone is not related to the spike shape. A low-pitched tone is played for a spike that matches no template, and a high-pitched tone for spikes that match templates numbered higher than 20. The short-cut key is Ctrl+U.

If you have no sound card the effect is operating system dependent (there may be no sound at all on some systems). With a sound card, there are limits on the maximum spike rate imposed by the operating system and the fact that sounds have to last much longer than the spikes for you to hear the tones!



### Scroll time view

If you depress this button, the associated time window scrolls to keep the current spike centred, if this is possible. The short-cut key for this is Ctrl+K.



### Automatic levels

This uses information collected from recent spikes to set the trigger and limit levels and to scale the data. This gives a good starting point for manual adjustment. The display scaling applies to both the data window and to the templates. The short-cut key is Home.

Details: Spike2 tracks the maximum excursion from zero in every displayed spike. If a new spike has a smaller maximum than the current maximum value, the current maximum is multiplied by 0.98. If the new spike has a larger maximum, it sets the new current maximum. The trigger levels are set at 0.7 and -0.7 of the current maximum. If you use this command in the New WaveMark dialog where you can have 4 horizontal cursors, the outer two levels are set to the current maximum and minus the current maximum. The Y scale is set to display from +1.4 times the current maximum to -1.4 times the current maximum.



### Scale data

These two buttons increase and decrease the display size of the waveforms and the templates. If you click on a button, the scale changes once. If you hold one down, the scale changes repeatedly until you release the button. You can also scale the data by clicking and dragging the y axis. The short-cut keys are Up arrow and Down arrow.



### Scroll data

These buttons scroll the data in the large window sideways. This changes the pre-trigger time, and is equivalent to changing the Pre-Trigger field of the Channel Parameters dialog for WaveMark data. If possible, the two triangles marking the start and end of the template region also scroll to preserve the template region. You can also click and drag the data x axis ticks to scroll the window or use the Left and Right arrow keys.



### Points

These buttons increase and decrease the number of points saved for each WaveMark. This is the same as the Channel Parameters dialog Points field. You can also use the PgUp and PgDn keys or click and drag the x axis numbers to scale the view around the 0 ms point. You should save as few points as possible to minimise the file size.



### Peak between cursors

Depress this button to display and use the limit cursors. Spikes with peak amplitudes at the trigger point that lie outside the limits are ignored. The short-cut key is Ctrl+H. You can use limits when creating new WaveMark channels from waveform data offline. If you have a Power1401 or a Micro1401-3, you can also use limits while sampling data. If you sample with a 1401 that cannot use limits, this button is hidden.



### Clear templates

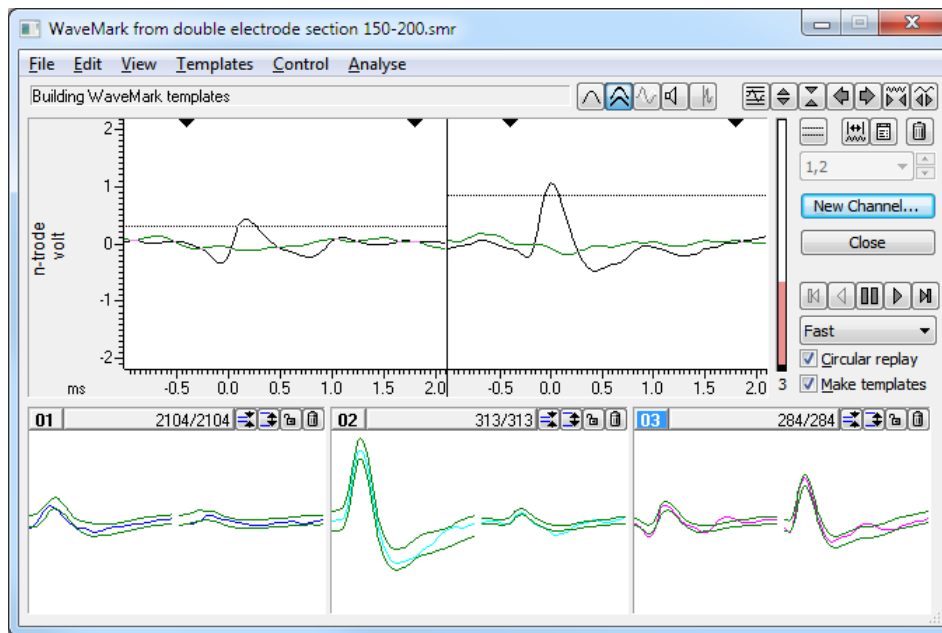
This button clears all templates (and any tentative templates not yet confirmed). Think of throwing the templates in the bin. The short-cut key is `Del`. You can clear individual templates using the clear button on the top bar of each template.



### Template parameters

This button opens a dialog that gives you control over template generation. To get the best use of the items in the dialog you will need to understand how templates are constructed and generated,. The short-cut key is `Ctrl+Enter`.

## Multiple traces



All the spike shape dialogs can handle WaveMark data with 1, 2 or 4 traces (single electrode, stereotrode and tetrode data). The documentation concentrates on the single trace case; this section explains the differences when you have multiple traces.

The data display is divided up horizontally so that each trace has its own rectangular area. The dialogs that allow you to set trigger and limit levels have independent horizontal cursors for each trace. The x axis and the triangles that mark the start and end of the template region are duplicated for each trace and are slaved together; if you drag the x axis or the template markers in one trace region, all traces change.

The templates are also divided up horizontally, however you can choose to overdraw the traces (perhaps to compare latencies) with the **View** menu **Overlay templates** command.

Spike sorting and template matching work in much the same way for multiple traces as for a single trace. The template parameters apply to each trace and all traces must meet the matching criteria for a spike to match a template. Because of this, you may need to relax some of the matching criteria compared to spike sorting with a single trace.

### Forming n-trode data from waveform channels

In addition to sampling data with 2 or 4 traces, you can create WaveMark data from 2 or 4 existing waveform channels that have the same sampling rate. To do this select the channels in the time view that you wish to convert by clicking on the channel number, then use the **Analysis** menu **New n-trode** command or right click on one of the selected channels and select the **New Stereotrode** or **New Tetrode** command. This will open the **New WaveMark** dialog with the selected channels set as the data source

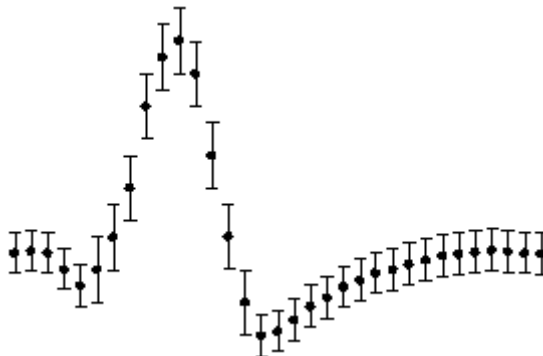
In the dialog, the channel selector is disabled and displays the list of channels that correspond to the traces. Channels are used in the order that you select them.

## Template formation

A template is stored as a series of data points. Each point has a width and a minimum width associated with it. The width represents the expected error in a signal that matches the template at that point. Spikes match a template if more than a certain percentage of the points in a spike fall within the template. It is easy to calculate an appropriate template width when you have a population of spikes of the same class. The problem comes when you have a single spike that starts a new template. What width do you give each point?

We set the initial template width to a percentage of the maximum positive or negative amplitude of the spike. The percentage to use depends on the data, 35% is a reasonable first guess. This value also sets the minimum allowed spike width. Experience shows that unless a minimum width is set, the template width tends to reduce as we only add matching spike. We also limit the sum of the differences between a spike and the mean template.

The template width is twice the mean distance between the template and the spikes that created it, but is not allowed to become less than the minimum width. If the variation in spike shape around the mean were normally distributed, you would expect 80% of the data in a matching spike to lie within the template.



When forming templates, the *Minimum percentage of points in template* rule applies, meaning that we do not consider a template if less than the user-specified percentage of data points lie within the template boundaries (defined as the mean template plus or minus the width at each point). This rule can be turned off when not forming templates and is turned off when forming templates if all provisional template space is exhausted.

The template building algorithm works as follows:

1. The first spike forms the first template with the template width estimated. This is a provisional template. Provisional templates are not displayed.
2. Each new spike is compared against each template in turn to see if sufficient points fall within the template. If there are any *confirmed* templates, these are considered first, and if a match is found, the provisional templates are not considered.
3. If a spike could only belong to one template, it is added to it. Otherwise, the spike is added to the template that is the minimum distance from the spike. Distance is defined as the sum of the differences between the spike and the template.
4. If a spike belongs to no templates, a new provisional template is created. If we have run out of provisional templates, we remove the *Minimum percentage of points in template* rule to give spikes more chance to match one of the provisional templates.
5. Adding a spike to a template changes the template shape and width. Once a pre-set number of spikes have been added to a template, it is checked against existing confirmed templates to prevent generation of two templates for the same data. If no match is found, the template is promoted to confirmed. If a match is found, the provisional template is merged with the confirmed one unless the confirmed template is locked, in which case a new confirmed template is created *with the same code as the one it matched*. Each time a spike is added to a provisional template, the provisional template *decay count* is reset.
6. Each time a spike is added to a template, all provisional templates have their decay count reduced by 1. If any decay count reaches 0, that template has 1 spike removed from its spike count and the decay count is reset. If the number of spikes in a provisional template reaches 0, the provisional template is deleted.

Step 2 is complex. To compare a spike against a template we shift it up to 2 sample points in either direction to find the best alignment point. Shifts of a fraction of a point are done by interpolation. If you have enabled amplitude scaling, the spike is scaled to make the area under the spike equal to the area under the template (limited by the maximum scaling allowed). Amplitude scaling is only allowed for confirmed templates. As interpolation is expensive, templates also store information to decide if it is worth attempting shifting to fit a spike to a template.

## Template settings dialog

The dialog controls the formation of new spike templates and how spike match templates. Each channel has an independent set of parameters. If you use the **Channel** control in the parent spike shape dialog to change data channel, the template parameters change to match the values for the new channel; changes made since the last **Apply** or **Copy to All** will be lost. The buttons at the bottom of the dialog are:

### *Copy to All*

Use this button when you have multiple channels of WaveMark data and you wish to apply the current template parameters to all of the channels. The action is the same as **Apply**, but it sets the value for all channels, not just for the current channel.

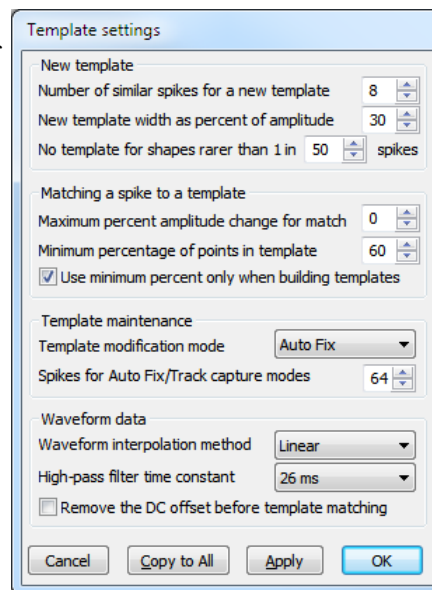
### *Apply and OK*

These do the same thing, but **OK** closes the dialog and **Apply** leaves it open. The settings you edit in the dialog make no difference to the template formation process unless you use one of the **Copy to All**, **Apply** or **OK** buttons. You cannot use these buttons if any field in the dialog holds invalid data.

### *Cancel*

This button closes the dialog without applying new changes. Changes made by the **Apply** or **Copy to All** buttons are preserved.

The dialog has 4 areas:



## New Template

This controls the creation of new templates.

### *Number of spikes for a new template*

This sets the number of spikes at which a provisional template is promoted to a real one. We find that 8 is a reasonable starting value.

### *New template width as a percentage of amplitude*

This is the percentage of the spike amplitude to give the initial (and minimum) template width. As spikes are added to the template, the width changes to represent the variation in amplitude of the spikes in the template. The maximum template width is set to 4 times the minimum width. A value of 30% is a reasonable starting value.

### *No template for shapes rarer than 1 in n spikes*

If this is  $n$ , this is roughly the same as saying that you are not interested in spike classes which occur less often than once every  $n$  total spikes. If you want to keep all spikes as potential templates you should set this to a large number. We find that 50 is a reasonable starting value. When discussing the template formation algorithm, we call this the *decay count* as provisional templates can be thought of as decaying if they do not get spikes added to them at reasonable intervals.

## Matching a spike to a template

The items in this group are used when comparing a spike with the existing templates to determine if the spike is the same or should start a new provisional shape. Unless amplitude scaling is enabled, to avoid wasting time interpolating spike shapes for data that could never fit a template, if the error between a spike and a template exceeds a value calculated from the template, the spike is rejected before applying the following:

### *Maximum percent amplitude change for match*

Spike2 will scale spikes up or down by up to this percentage to make the area under the spike the same as the area under each target template. This is very useful if you have spikes that maintain shape but change amplitude. Do not set this non-zero on-line unless you need to as it slows down the template matching process.

---

The maximum change permitted is 100%, which allows a spike to be doubled or halved in amplitude. Set this to 0 unless you need it. The value you set depends on the amplitude variability of your spikes; 25% is a reasonable starting value.

#### *Minimum percentage of points in template*

The percentage of a spike that must lie within a template for a match. If more than one template passes this test, spikes are matched to the template with the smallest error between the mean template and the (scaled) spike. 60% is a reasonable starting value. With amplitude scaling on, you may want to increase this to 70% or more.

#### *Use minimum percentage only when building templates*

The template width is most useful in the set up phase when we are looking carefully at differences between spikes. Once templates are established it can sometimes be better to match to the template with the smallest error and ignore the width. If you check this field, this sets the percentage of points that must lie in the template to 0% unless you are building templates.

## **Template maintenance**

This group of fields determines how the shape of a template changes as more spikes are added to it. On-line, the template shape is fixed in all modes apart from **Track**. From our experience, **Add All** and **Auto Fix** are the most useful modes.

#### *Template modification mode*

- Add All** All spikes that fit the template are added and modify the template. The effect of each spike becomes smaller as the number of matched spikes gets larger.
- Auto Fix** The template is fixed once a set number of spikes have been added. If you have several similar spikes, using **Auto Fix** after a fairly small number of spikes can stop a template gradually changing shape and becoming the same as another template.
- Track** The template shape tracks the spikes. The contribution of each spike to the template decays as more spikes are added. This is only really useful for slow changes in spike shape and always brings with it the danger that all your shapes will merge together. It also slows down on-line spike classification.

#### *Spikes for Auto Fix/Track capture modes*

This sets the number of spikes for the previous field in **Auto Fix** and **Track** modes. In **Track** mode, the smaller the number, the more rapidly the template shape changes.

## **Waveform data**

The final group of fields control how the raw waveform data is processed into spikes:

#### *Waveform interpolation method*

Spike waveforms are shifted by fractions of a sampling interval to align them with templates using linear, parabolic or cubic spline interpolation. Linear interpolation is the fastest, cubic spline interpolation is the slowest. The on-line 1401 code always uses linear interpolation for speed reasons. For off-line work, speed of computation does not matter.

#### *High-pass filter time constant*

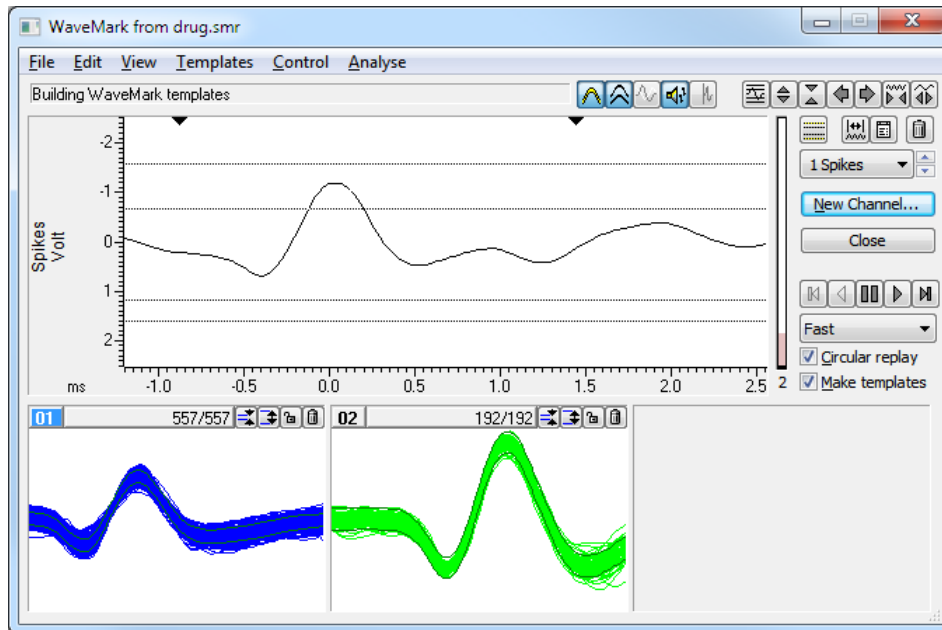
Use this to remove baseline drift. Set this to a few times the width of the spikes. Do not set the value too low or it will significantly change the spike shapes. If your signal has abrupt baseline changes, you may get better results with the DC offset option.

#### *Remove the DC offset before template matching*

Check this field to subtract the mean level from each spike before matching. This effects template formation and matching, not the saved data. Unless your baseline has sudden DC shifts, it is usually better to use the high-pass filter to follow signals that drift. There is a small time penalty for using DC offset removal on-line.

## Off-line template formation

The Analysis menu **New WaveMark...** command extracts WaveMark data from waveform channels (and also from existing WaveMark data). If you wish to edit WaveMark data, see the *Off-line template editing* section.



This window is almost identical to the on-line template setup window. There are more menu items and buttons in the control bar, some button labels have changed and there are new controls to set the spike replay rate and you can print and copy template data. You can use the horizontal cursors to set the trigger levels and limit spike amplitude.

The program scans the data in the channel set by the channel control (wrapping round to the start of the data when it reaches the end if **Circular replay** is checked), and displays triggered and non-triggered events, as if the data were being sampled.

### Replay rate

This control sets the maximum number of spikes per second to process when running continuously. With **Fast** selected, the spikes are processed as fast as possible; you can also set this by pressing the **F** key (**F** for Fast). With **Real time** selected, spikes are replayed no faster than real time; you can set this with **R** for Real time. You can also select slower rates in Hz; the **S** for Slow key selects a medium-slow rate.

### New Channel...

Once you are satisfied with your templates, click the **New Channel...** button to open a dialog in which you can select the channel to write to and the new data type.

### Time range

This button opens a dialog in which you set the time range of data to process. You can either type in the start and end times, or select values from the drop down list. When you open the window for the first time, the time range is set to the whole file.

### Cursor 0

While this window is open, cursor 0 is added to the time view associated with the data file. This cursor marks the position of the current spike. When paused, you can drag this cursor to a new position. The special cursor does not appear in Cursor windows. The **Ctrl+J** key combination jumps Cursor 0 to the start of the time range.



## Circular replay

This check box enables circular continuous replay. If you clear this box, continuous replay stops when it reaches the end of the time range set for analysis. The short-cut key for this is `Ctrl+Q`.

## Scroll time view

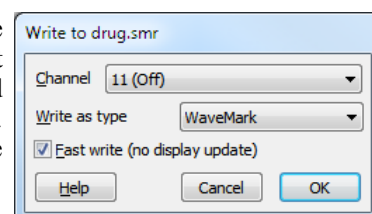
If you depress this button, the associated time window scrolls to keep the current spike centred, if this is possible. The short-cut key for this is `Ctrl+K`.

## Marker filter

The Analysis menu Marker filter command (short-cut key `Ctrl+F`) is enabled when the source channel holds WaveMark data. It opens the Marker Filter dialog, or brings it to the front if it is already open. Whenever you change channel to a channel holding WaveMark data with the Marker filter dialog open, the Marker filter dialog automatically changes channel to match.

## Create a New spike channel

This dialog opens when you click the New Channel button or select the New Channel command from the Spike shape dialog File menu. You must select the channel to write to; the dialog will offer you the lowest numbered unused disk channel, but you can choose to overwrite an existing channel. The Fast write check box suppresses display updates whilst generating the new channel, to save time. You can also choose the channel type to write the data in (select WaveMark if you are unsure):



**Event-, Event+** If you use this format, only the time stamp for each detected event is saved. The new channel has no template matching or waveform information.

**Marker** This saves the template matching information and the time stamp, but does not save any waveform information.

**WaveMark** You will usually use this format to save a time stamp, a marker code and the waveform for each detected event.

When you choose OK, the selected time range of data is analysed, and any events that cross the trigger levels are written to the new channel in the selected format. Options that change the number of points saved per WaveMark are disabled during analysis.

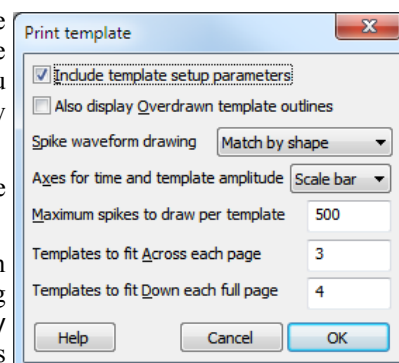
The title of the new channel is set to `nw-c` where `c` is the original channel number. The channel comment for the new channel is set to indicate the source of the data.

## Print templates and Copy

The File menu Print... command (short-cut key `Ctrl+P`) opens the Print template dialog. The output format depends on the printer page size (you can set the print margins in the main application File menu Page setup dialog). You can choose how many templates to draw across and down each page.

You can include the template setup parameters, and also an extra picture showing all of the template outlines overdrawn for easy comparison.

The Spike waveform drawing field chooses how spikes are included in the result. Set it to None for no spikes or to the method for selecting which spikes are drawn with a template. You can choose Match by shape or Match by code (only in the Edit WaveMark dialog). Spikes are drawn in the best-fit position to the template. You can limit the number of spikes to draw in each template; some printers cannot cope with huge numbers of lines in the printed output.

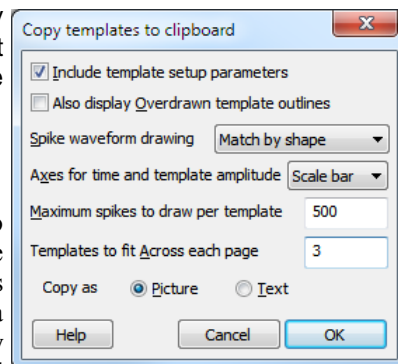


The Axes for time and template amplitude field lets you choose between None (no axes drawn), Scale bar (an indication of size) and Full (a full axis with big and small ticks). The y axis range is the same as the y axis range in the Spike shape dialog. The example image below is using a scale bar; this is the default setting.

If you have set a time range to analyse, the overdraw spikes are taken from the time range. If you have not set a time range, the spikes are taken from the start of the file.

### Copy to clipboard

The Edit menu Copy command (short-cut key **Ctrl+C**) opens the Copy templates to clipboard dialog, which is very similar to the Print templates dialog. There are radio buttons to choose between Picture and Text output.

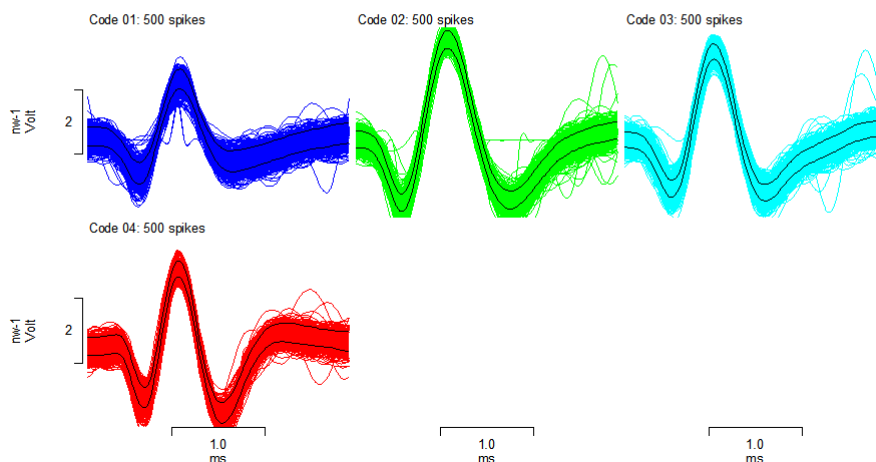


### Typical clipboard output

Picture output is in Enhanced Metafile format and can be pasted into documents in most drawing programs and many word processors. Some bitmap-editing programs (for example, the Paint program that comes as part of Windows XP) will accept this format and convert it into a bitmap. If you want to import data into a drawing program you may need to restrict the number of spikes displayed per template as not all drawing programs cannot cope with a large number of vectors in an image.

Templates from drug.smr channel 6

|                                             |       |                                         |         |
|---------------------------------------------|-------|-----------------------------------------|---------|
| Number of templates                         | 4     | Maximum % amplitude change for match    | 0       |
| Points in each template                     | 36    | Minimum % of points in template         | 60      |
| Traces in each template                     | 1     | Use minimum % only if making templates  | Yes     |
| Template start offset to spike peak         | 14    | Template modification mode              | Add all |
| Sample rate in Hz                           | 12500 | Spikes for Auto Fix/Track capture modes | 64      |
| Number of similar spikes for a new template | 8     | Input filter time constant              | 41 ms   |
| New template width as % of amplitude        | 20    | Remove DC before matching to template   | No      |
| No template for shapes rarer than 1 in      | 50    |                                         |         |



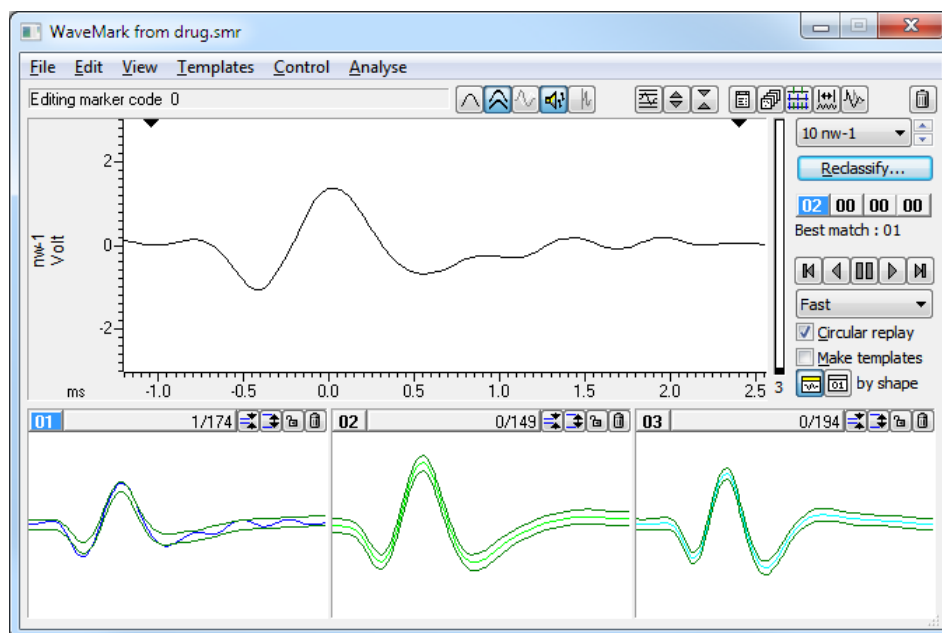
Text output optionally includes the setup parameters. The templates are output in vertical columns separated by Tab characters. There is one column or two columns per template, depending on the template shape display mode (mean templates or upper and lower limits). The first row of template output holds column titles, in quotation marks. The output is designed to paste easily into spreadsheet programs. This example is showing the upper and lower template levels (it is displayed in a table to demonstrate the structure of the output).

|                                               |          |          |          |          |           |          |          |    |  |
|-----------------------------------------------|----------|----------|----------|----------|-----------|----------|----------|----|--|
| "Number of templates"                         |          |          |          |          | 4         |          |          |    |  |
| "Points in each template"                     |          |          |          |          | 36        |          |          |    |  |
| "Traces in each template"                     |          |          |          |          | 1         |          |          |    |  |
| "Template start offset to spike peak"         |          |          |          |          | 14        |          |          |    |  |
| "Sample rate in Hz"                           |          |          |          |          | 12500     |          |          |    |  |
| "Number of similar spikes for a new template" |          |          |          |          | 8         |          |          |    |  |
| "New template width as % of amplitude"        |          |          |          |          | 20        |          |          |    |  |
| "No template for shapes rarer than 1 in"      |          |          |          |          | 50        |          |          |    |  |
| "Maximum % amplitude change for match"        |          |          |          |          | 0         |          |          |    |  |
| "Minimum % of points in template"             |          |          |          |          | 60        |          |          |    |  |
| "Use minimum % only when making templates"    |          |          |          |          | "Yes"     |          |          |    |  |
| "Template modification mode"                  |          |          |          |          | "Add all" |          |          |    |  |
| "Spikes for Auto Fix/Track capture modes"     |          |          |          |          | 64        |          |          |    |  |
| "Input filter time constant"                  |          |          |          |          | "41 ms"   |          |          |    |  |
| "Remove DC before matching to template"       |          |          |          |          | "No"      |          |          |    |  |
| "Code                                         | 01"Code  | 01"Code  | 02"Code  | 02"Code  | 03"Code   | 03"Code  | 04"Code  | 04 |  |
| upper"                                        | lower"   | upper"   | lower"   | upper"   | lower"    | upper"   | lower"   |    |  |
| 0.263062                                      | -0.12420 | 0.189056 | -0.15548 | 0.169983 | -0.14434  | 0.238342 | -0.13397 |    |  |

|           |          |          |          |           |          |          |           |
|-----------|----------|----------|----------|-----------|----------|----------|-----------|
| 0.266724  | -0.12085 | 0.170135 | -0.16952 | 0.154877  | -0.15945 | 0.236664 | -0.13412  |
| 0.255432  | -0.12878 | 0.120239 | -0.21911 | 0.114899  | -0.20065 | 0.249329 | -0.13092  |
| 0.206909  | -0.16540 | -0.05386 | -0.39321 | -0.008544 | -0.32226 | 0.2771   | -0.110779 |
| 0.08255   | -0.28121 | -0.41549 | -0.75485 | -0.258636 | -0.58090 | 0.294037 | -0.093841 |
| -0.130463 | -0.50918 | -0.85144 | -1.20117 | -0.576324 | -0.91323 | 0.147552 | -0.237885 |
| -0.359955 | -0.77896 | -1.09451 | -1.45981 | -0.804749 | -1.1499  | -0.25009 | -0.673676 |
| -0.461731 | -0.90942 | -0.94528 | -1.30295 | -0.761566 | -1.08292 | -0.69107 | -1.12625  |
| -0.326385 | -0.73837 | -0.38452 | -0.72601 | -0.323792 | -0.63781 | -0.76080 | -1.15631  |
| 0.0418091 | -0.27069 | 0.437317 | 0.097961 | 0.41214   | 0.098419 | -0.25512 | -0.617981 |
| 0.57251   | 0.313416 | 1.27914  | 0.939789 | 1.20239   | 0.888672 | 0.605011 | 0.264435  |
| 1.11221   | 0.782623 | 1.91879  | 1.57883  | 1.77475   | 1.46072  | 1.39893  | 1.0788    |
| 1.41571   | 1.02417  | 2.21878  | 1.866    | 1.96671   | 1.65298  | 1.79184  | 1.46744   |
| 1.37802   | 0.992279 | 2.14111  | 1.79016  | 1.74911   | 1.43539  | 1.63803  | 1.30844   |
| 1.04614   | 0.726929 | 1.74225  | 1.40228  | 1.21918   | 0.905457 | 1.02386  | 0.713501  |
| 0.607147  | 0.328827 | 1.15738  | 0.818024 | 0.542755  | 0.229034 | 0.198975 | -0.076904 |
| 0.232697  | -0.06637 | 0.500793 | 0.161438 | -0.102997 | -0.41671 | -0.52581 | -0.827942 |
| -0.027160 | -0.37445 | -0.09979 | -0.43914 | -0.590973 | -0.90805 | -0.96420 | -1.33621  |
| -0.164642 | -0.55648 | -0.57418 | -0.91354 | -0.852814 | -1.18149 | -1.10382 | -1.51031  |
| -0.215607 | -0.63247 | -0.88302 | -1.22665 | -0.906372 | -1.24695 | -0.98861 | -1.39847  |

## Off-line template editing

The Analysis menu **Edit WaveMark...** command opens a dialog in which you reclassify data in a WaveMark channel. This dialog has its own menu system that includes links to Principal Component Analysis and clustering. If the selected channel has a marker filter set, you view and edit only those events that match the filter specification. The window controls are the same as for *Off-line template formation*, except that you cannot change the number of points in the main window and there are no horizontal cursors (use **New WaveMark** if you need trigger levels).



### Marker codes **02 00 00 00**

These four codes show the four marker codes of the current spike. Normally only the first code is used for spike classification and the others are 00. If you click on a code, it becomes highlighted and sets the code that is changed by reclassification. If you double-click a code a dialog box opens in which you can edit the code. The **Best match** code is the code of the template that best matches the current spike or 00 if no template matches. If you use this dialog on-line only the leftmost button is enabled.

### Template formation method by shape

You can build templates by shape or by code. If you build by shape, templates are formed and spikes are sorted based on shapes; prior classification is ignored. If you build by code, the shapes are ignored, and the classification is purely by the code highlighted in the **Marker codes**. Templates are built automatically when the play controls are set to play forward or backward through the data and when the **Make templates** box is checked.

### Manual reclassification

When replay is stopped, you can reclassify a spike by dragging the raw data and dropping it on a template or by double clicking a Marker code and editing. You can also reclassify by pressing keys 0 to 9 to give a spike template marker codes 00 though 09. If you use keys 1 through 9 and no template exists with the code, a new template with the code is created based on the current spike.

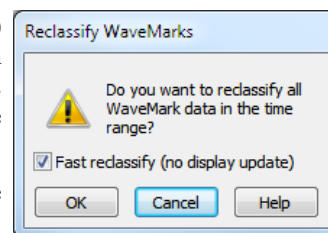
### Automatic reclassification Reclassify...

Click the Reclassify... button to open the Reclassify WaveMark dialog to reclassify all the spikes in the current time range based on the templates.

## Reclassify WaveMarks

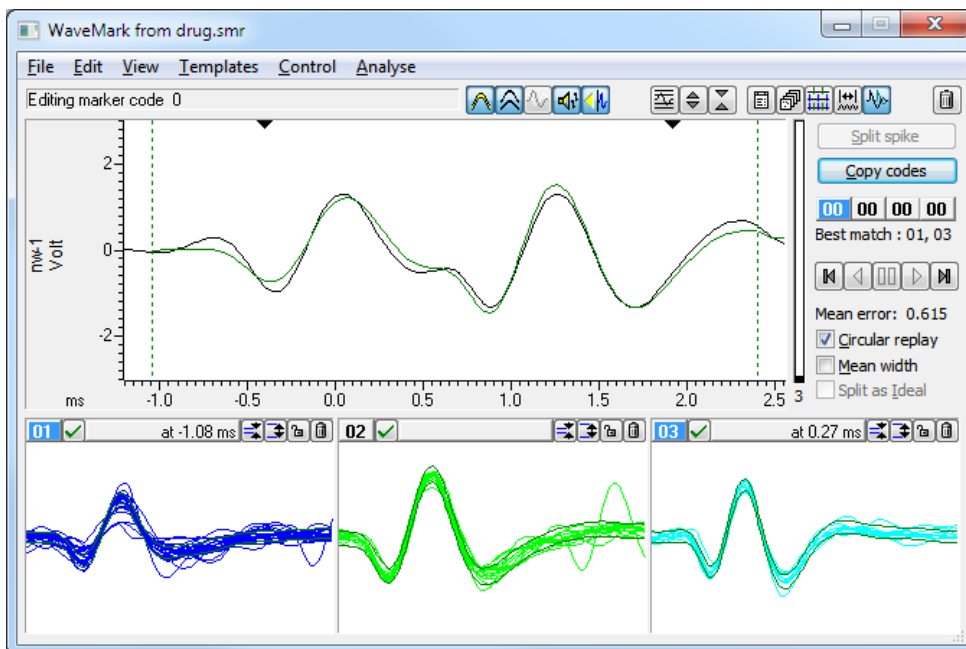
This option reclassifies all the events (or filtered events if a marker filter is set) on the channel in the time range by matching them to the templates. You can set fast reclassification (no display) or a slower mode where each event is drawn in the data display area as it is reclassified. You can stop the reclassification before all spikes have been scanned, but you cannot undo it.

*Tip:* clear the Make templates check box before reclassifying to stop Spike creating extra templates and modifying existing ones.



## Collision Analysis Mode

If a channel contains more than one class of spikes, and the spikes are independent, it is inevitable that there will be collisions. A simplistic analysis shows that if two independent spikes have widths  $w_1$  and  $w_2$  and mean firing rates  $r_1$  and  $r_2$ , we would expect  $r_1 * r_2 * (w_1 + w_2)$  collisions per second. If both rates are 10 Hz and both widths are 2.5 ms, you would expect 1 collision every 2 seconds. If we can assume that the resulting waveform is the sum of two spike waveforms, and that these waveforms are similar to the established templates, we can search for the template pair whose sum is most similar to the collision.



To enter or leave Collision Analysis Mode, use the Analysis menu command or click on the collision analysis button at the right-hand end of the toolbar. You will notice that some of the controls on the right of the window change, as does the main data display:

## Main display

The main display shows the current spike overlaid with the best match combination of templates. The dashed vertical lines show the start and end of the template-forming region. The two black triangles mark the start and end of the *Important area*.

## Important area

You can change the important area by clicking and dragging the black triangles. If the area is smaller than the templates (the usual case), it is always included in the best match. If it is more than twice the size of the templates, the best match area will lie within it. In an intermediate case, it marks the area we would like to match and some or all of the best match will lie in the region.

## Template display area

The button to the right of the template code excludes the template from the matching process. You might want to do this if a collision is too close to another spike of the same class. You can exclude all but 1 template. Templates used to make a best match have the template code highlighted and list the offset in milliseconds into the main display at which the template starts.

## Best match

This lists the template code or codes that were used to make the best match trace in the data display. The code for the earlier template is listed first. Templates are allowed to overlap the start and end of the displayed spike when making a match, but at least half the template points must be used in the match.

## Copy codes

Click this button to copy the *Best match* codes to the current spike. The first code is applied to the currently selected marker code. If there are two codes, the second is applied to the next marker code to the right (wrapping round to the first if necessary).

## Mean error

This field displays the mean square error per point. The error is scaled by either the template width at each point, or by the mean template width over all the templates. See the following description of *Mean width*.

## Mean width

If you check this box, the errors are scaled by the mean width of all the templates (including excluded templates). This gives all points the same importance when calculating the best template combination. Think of this as "least squares" fitting. If you do not check the box, the template width is used point by point to scale the error. This gives points with a small template width more importance than points with a larger template width. Think of this as "Chi-squared" fitting.

If your templates have a wide variety of widths, you may get better results with this box checked (or at least results that are more what you expect).

## Split spike

This button is enabled if the spike source is a memory channel and the current spike is worth separating. Worth separating means that the spike matches two templates, or it matches a single template that doesn't overlap the x axis zero (it matches a template that lies entirely before or after the peak in the raw data).

This replaces the current spike with spikes aligned to the best match templates. How the replacements are calculated depends on the *Split as Ideal* check box. Each matching template generates one output spike. If the best match doesn't include the x axis zero, the original spike (less the best match waveform) is also preserved and *Split as Ideal* is assumed to be checked.

## Split as Ideal

If this is checked (or assumed checked), the replacements waveforms are the template shapes. If this is not checked, the replacements share half the difference between the original data and the best match so that the sum of the two spikes recreates the original where the two created spikes overlap. In both cases, any data that is not available from the original data is set to zero.

## Caveats

It is important that the templates are centred vertically on zero. If this is not the case, adding templates will create offsets, and the results will not be useful. Likewise, each template should start and end around zero. If they don't, you probably have not selected enough data when forming the templates. If templates start and/or end with a substantial non-zero value, the non-zero ends cause discontinuities in the best match waveform. You must also have sampled fast enough that the change between two consecutive samples is not so large that the matching algorithm fails.

This method will work best in cases where you have a small number of well-defined templates. If you have a large number of indistinct templates, you will likely be able to match just about any shape, but the results will be meaningless. We suggest that you only use this method where there is an obvious collision and that you treat the results with great caution. Look at the firing patterns of the spike classes and check that the resolved collisions fit the patterns. It can be revealing to exclude a template from a collision to see how close the analysis can get with another pair of spikes.

## The matching algorithm

The best match is found by an exhaustive search of all possible pairs of templates at all alignments against the spike data identified by the *Important area*. The searched area may be further restricted by the rule that at least half the points in each template must overlap with spike data points. Templates are combined by adding the mean template shapes and calculating a width. If *Mean width* is checked, the width is the same at all points and is the average width of all templates. If *Mean width* is not checked, non-overlapped areas of the combination copy the original width and overlapped areas use the RMS (root mean square) of the two widths.

For each pair of templates, there are two variables to consider: the offset of the second template with respect to the first, and the offset of the combination into the data. We find the best value of both variables using integral point shifts, then improve the best value by finding the best fractional position of both variables by interpolation.

The match criterion is the mean squared error per compared point. At each point, the error is the difference between the spike and the template combination divided by a width. The width is either the template width at that point, or is the average template width over all templates. The average width includes excluded templates so that errors are comparable when templates are excluded.

## Analyse menu commands

The Edit WaveMark dialog **Analyse** menu contains a range of commands for cluster and Principal Component Analysis, plus commands to manipulate the events that you work with:

### Principal Component Analysis

This command (short-cut key `Ctrl+A`) initiates PCA analysis of all the spikes in the current time range using the same waveform area as selected for the templates. This is dealt with in the Clustering section.

### Cluster on measurements

This command (short-cut key `Ctrl+Shift+A`) opens the Cluster setup dialog in which you select measurements to take from each spike in the time range for cluster analysis. This is covered in the Clustering section.

### Marker filter

The Marker filter command (short-cut key `Ctrl+F`) opens the marker filter dialog, or brings it to the front if it is already open. Whenever you change channel to a channel holding WaveMark data with the Marker filter dialog open, the Marker filter dialog automatically changes channel to match.



## Duplicate

This command (short-cut key `Ctrl+D`) creates a duplicate data channel in the time view for each template code in the dialog. It is disabled if the current channel is a duplicate channel. It does the following:

1. All duplicates of the current channel are deleted (with no warning) from the time view and from any duplicates of the time view. Any processing that depended on the duplicated channels is cancelled.
2. Spike2 counts how many different template codes you have defined. For example, if you have created four templates with codes 01, 02, 02 and 03, this is counted as 3 different codes.
3. For each template code, Spike2 generates a duplicate channel with the marker filter set to display only spikes that match the template code. The marker filter mask used is the same as the code highlighted in the Marker codes (usually the first mask is used). The channel title of each duplicate is set to "title-`nn`", where `title` is the original channel title (shortened to 6 characters if too long) and `nn` is the template code in hexadecimal. The new channel is positioned next to the original channel in the time window. It is placed above the original channel unless the Edit menu preferences have reversed the channel order, in which case it is placed below the original.

The assumption is that you have already, or are about to, classify the spikes in the channel based on the templates in the dialog. If you have duplicated the time view, the new duplicate channels exist in all views, but are only displayed in the view that is linked to the Edit WaveMark dialog. The original channel will remain, unchanged, in all the views.

## Set Codes

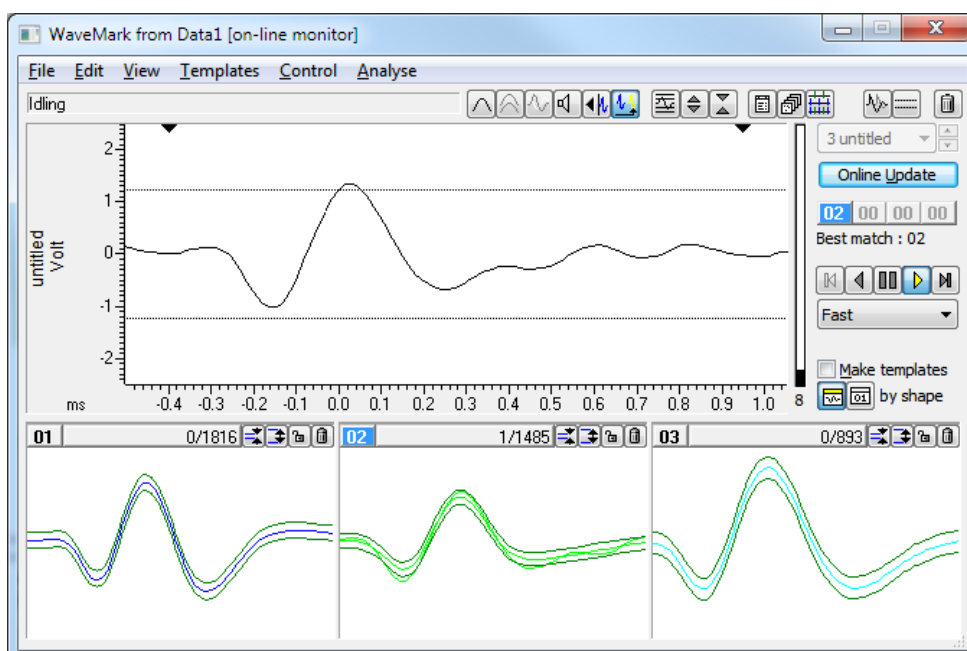
This command (short-cut key `Ctrl+E`) opens the Set Marker Codes dialog. You can use this to give all the displayed spikes in a channel the same marker code. For example, if you want to change all the spikes on channel 2 with codes 04, 05 and 09 to have code 03:

1. Open the Marker Filter dialog and set the filter for channel 2 to display only codes 04, 05 and 09.
2. Click the Set Codes button and set the first code to 03 and click the Set button.

## Collision Analysis Mode

This command (short-cut key `L`) toggles between normal editing mode and collision analysis mode. Normal editing mode attempts to match each spike to a single template. Collision analysis mode attempts to match the spike to a collision of two templates.

## On line template monitoring



The Analysis menu Edit WaveMark... command can be used during data sampling if there are any WaveMark channels. The window that opens is very similar to the off-line template editing window; the time range and reclassify buttons are hidden, only the first of the Marker code buttons is enabled, the Reclassify... button is renamed to Online update and there is a new button in the control bar. If there are no spikes to display the view shows the untriggered waveform so you can adjust the horizontal cursors.

 **At end mode**

When this button is down, all spikes are taken from the end of the file, the horizontal cursors that set the trigger levels appear and controls that cannot be used are hidden. In this state, you monitor the spikes as they are classified by the 1401. If there are no spikes, the data display shows recent waveform data so that you can adjust the trigger levels.

With the button up, the window behaves very similarly to the normal off-line template editing except that the Reclassify command is not available. Once sampling finishes the at end of file button and horizontal cursors vanish, the Reclassify... button appears and the normal off-line editing behavior is restored.

**Online update**

During sampling, the CED 1401 interface classifies spikes based on the templates and parameters that were last loaded to it. If you generate more spike templates, change the existing templates, or edit the template parameters this will make no difference to the 1401 unless you click the Online update button. This deletes all templates stored in the 1401 for the current channel and replaces them with the templates displayed in the window. The trigger levels are updated dynamically; you do not need to click the Online update button to change the trigger levels.

**On-line and Off-line templates**

We save two types of template in the sampling configuration and in the resource files associated with a data file: *on-line* templates and *off-line* templates. Spike2 saves on-line templates when you use the On-line template setup dialog and when you click the Online update button while monitoring spikes during sampling. Off-line templates are saved when you change channel or close the spike shape dialog.

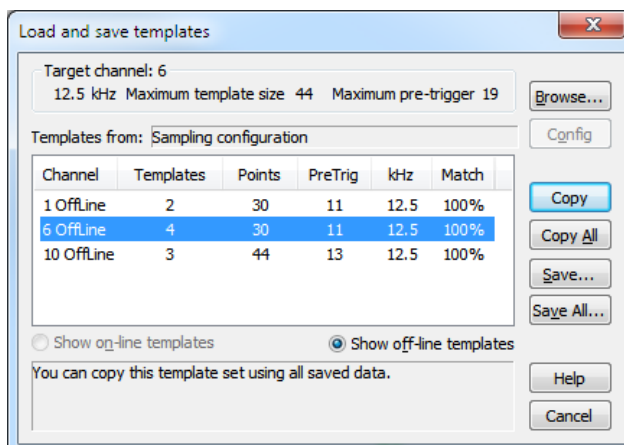
When you change channel, Spike2 loads templates from the saved on-line templates if you are sampling and from the saved off-line templates if you are not sampling. This is so that the templates you see during sampling are as close as possible to the templates that the 1401 uses. Off-line templates are loaded from the resource file associated with the data file. If none are found or there is no resource file, off-line templates are loaded from the sampling configuration.

In summary, on-line templates are the last set of templates copied to the 1401 during sampling. Off-line templates are the last set of templates displayed in a template dialog.

**Load Save templates**

You can save templates and load templates into the current channel or into all matching channels with the Load/Save... button. Spike2 saves templates in the sampling configuration and in the resource files associated with data files. Whenever you work on templates in a data file, for example data1.smr, both the associated resource file data1.s2r and the sampling configuration are updated to hold the latest set of templates. You can save the sampling configuration to disk in the File menu.

The box at the top of the dialog shows the sample rate and template size limits of the current target channel. This is the channel that is updated by the Copy command.



The box in the centre of the dialog lists sets of templates from the Sampling Configuration or loaded from a resource or configuration file. When you open the dialog, Spike2 shows you templates in the Sampling



Configuration. The columns are:

|                  |                                                                                                                                                                                                                                                                                 |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Channel</b>   | Identifies the channel from which the templates originated.                                                                                                                                                                                                                     |
| <b>Templates</b> | The number of templates stored for this channel. Channels with no templates are not listed. All templates for a channel have the same sampling rate, points and number of pre-trigger points.                                                                                   |
| <b>Points</b>    | The number of data points in each template.                                                                                                                                                                                                                                     |
| <b>PreTrig</b>   | The number of data points in each template that lie before the peak or trough used as a trigger. A negative value means that the template starts this many points after the trigger point.                                                                                      |
| <b>kHz</b>       | The sample rate of the template source channel. This need not match the target channel; Spike2 uses cubic spline interpolation to compensate for sample rate differences. Templates generated before Spike2 version 3.15 did not include a sampling rate and 25 kHz is assumed. |
| <b>Match</b>     | This columns displays how much of each template shape would be used if this set of templates were copied to the target channel.                                                                                                                                                 |

### Browse... and Config

Use the **Browse...** button to list templates in resource (.s2r) or sampling configuration (.s2c) files. The **Config** button displays templates stored in the sampling configuration.

### Show on/off-line templates

Templates copied to the 1401 during sampling (on-line templates) are saved separately from the templates displayed in the template dialogs (off-line templates); you choose which to list with **Show on/off-line templates**. The **Copy All** and **Save All** commands operate on the list selected by this option.

### Copy

The **Copy** button copies the selected set of templates into the target channel and closes the dialog. You can copy a channel as long as **Match** is at least 10%. This overwrites any existing templates in the target channel.

### Copy All

For each channel in the channel control, Spike2 searches the list of templates for a matching channel number. If templates exist for the channel and they match the channel settings, they are copied from the list and become available for use.

### Save

This saves the selected set of templates in the list to a resource file (.s2r file extension). You are prompted to supply the name of a new file or to select an existing resource file. If the resource file already exists, any templates for the channel are replaced. If the resource file does not exist, the templates are written to a new file.

### Save All

This saves all the templates in the list to a resource file (.s2r file extension). You are prompted to supply the name of a new file or to select an existing resource file. If the resource file already exists, any existing templates for the channels in the list are replaced. If the resource file does not exist, the templates are written to a new file.

### Cancel

This button closes the dialog.

## Template storage

You use templates to classify WaveMark data, but they are not stored in data files. Instead, Spike2 stores templates in the following places:

- In the sampling configuration. You can always load the last set of templates you used on-line or displayed off-line in the current session from here. You must save the configuration to make templates persist after you close Spike2.

- In saved configuration files with the file extension `.s2c`. For example `Last.s2c` automatically saves the last configuration used for sampling. However, changes made in the template dialogs after sampling ends are not saved automatically.
- In resource files associated with data files. Each time you use a template dialog, any templates you create or change are saved in a resource file with the same name as the data file, but with the file extension `.s2r`. During sampling, there is no associated resource file. In this case, when you save the data file, Spike2 creates a resource file and copies the templates in the sampling configuration to it.
- In resource files created from the **Load and Save templates** dialog.

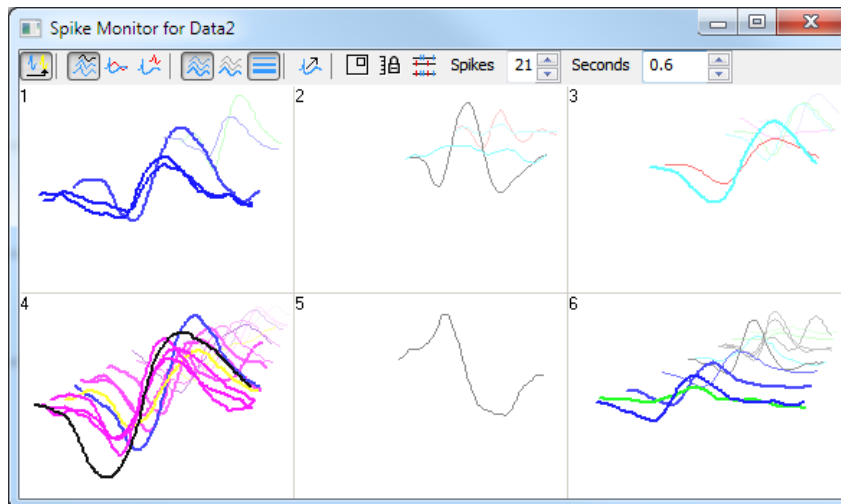
When you open a new data file for sampling, the template dialogs load up the last set of on-line templates from the sampling configuration. These on-line templates are updated each time you click the **Online update** button in the **Edit WaveMark** dialog. The off-line templates are updated to match the templates displayed in the template dialog. When you stop sampling and save the file, the templates are written to both the resource file associated with the data file and to the sampling configuration.

When you work on a data file from disk and open a template dialog, the off-line templates are loaded from the resource file associated with the data file. If none are found, off-line templates are loaded from the sampling configuration. Any changes you make are saved to both the resource file and to the sampling configuration as off-line templates.

By saving templates to both the resource file and to the sampling configuration, Spike2 makes it easy for you to work on a sequence of data files. For example, you might break a sampling session into several data files. On-line templates created for the first data file are automatically used for the second and subsequent files.

## Spike Monitor

The Spike Monitor window can be opened with the **View** menu **Spike Monitor** command or from the script language with `SMPopen()`. The main use of the window is to give an overview of the current spike activity on all channels during sampling. You can drag and resize the window to position it; the grid of cells organises itself automatically. Each cell displays the channel number at the top left and spikes that lie in a user-defined time range back from the current time. The *WaveMark background* field of the *Application colours* sets the cell background colour.



The window is linked to the **Edit WaveMark** dialog. Double-click a cell to open the **Edit WaveMark** dialog. Single-click a cell to select the corresponding channel in the **Edit WaveMark** dialog. The display is controlled by the toolbar at the top of the window. The toolbar commands are:

### At end

This button is enabled for on-line use and during rerun off-line. With the button down, spike data comes from the end of the file. With the button up, the data comes from the cursor 0 position in the associated time view. If you start to rerun a data file, this window automatically switches to *At end* mode, and cancels this mode when the rerun ends.

## Display mode

There are three display modes: *3D*, *2D* and *2D separate*. Spikes are positioned in the cell using two rectangles: a front rectangle that is anchored to the bottom of the cell and a back rectangle that is anchored to the top of the cell. You can display the rectangles and change their positions with the Rectangle command, described below. Normally the front rectangle is larger than the back rectangle. The display modes are:

### *3D*

Each spike is drawn in a rectangle that is positioned between the front and back rectangles depending on how old the spike is relative to the current time. New spikes appear in the front rectangle and then move to the back rectangle as time passes.

### *2D*

All spikes are drawn in the front rectangle; the back rectangle is not used.

### *2D separate*

The last spike is drawn in the front rectangle, the rest are drawn in the back rectangle.

## Colour fade

With this button down, the spikes colours fade into the background colour with time. A new spike is drawn in its normal colour. A spike that is half the user-defined time range old would draw in a colour that is half way between the normal colour and the background colour.

## Colour front only

With this button down, only the newest spike is drawn in its normal colour. All other spikes appear in a grey that contrasts with the background enough to be visible. This is particularly useful in the 2D drawing mode.

## Vary line thickness

Normally, all spikes are drawn with the same line thickness as data in a time view as set in the Edit menu Preferences. With this button down, the lines become thinner with time, with new spikes being drawn with a thick line. The line thickness for a new spike depends on the height of the cell. Use this option with caution; a thick line takes longer to draw than a thin one.

## Timed (smooth) updates

In *At end* mode, cells update when new spikes enter or leave the time range, which can lead to a jerky display. With this button down, cells update continuously, which can look much better, particularly in the 3D display mode. However, this option uses more system time especially if *Vary line thickness* is enabled. The continuous update rate is decreased when the Spike Monitor window is not the active window to make more time available to other windows.

## Show rectangles

Normally, the front and back rectangles that are used to position spikes are hidden. With this button down, the rectangles are drawn and can be moved by clicking and dragging with the mouse. The spikes are still displayed, but are drawn in grey so that the rectangles are clearly visible. To move a rectangle sideways, click in it and drag it. To size a rectangle, click in it to display two drag handles and then click on a drag handle to resize.

## Lock y axes

With this button up, the spikes in each cell self-scale. Each cell tracks the maximum amplitude it has seen. When a new spike that exceeds the current maximum amplitude is seen, it updates the maximum amplitude. This new maximum is held for two seconds, then it decays with time. With this button down, the current y axis scale is held.

## Duplicate channels

With this button up, duplicate channels are not displayed and any marker filters set for channels are ignored. With this button down, duplicate channels are displayed and only the spikes that meet the marker filter settings for each channel are displayed.

## Spikes

You can set the maximum number of spikes to display in each cell, in the range 1 to 40. The time taken to display the data depends on the number of spikes, especially if you enable *Vary line thickness*. You should set this to the minimum number that is useful.

## Seconds

This field sets the time back from the current time range over which to display spikes. It is normal to set this to a small number of seconds, but you may need to set a very short period if you are trying to visualise burst characteristics in the 3D display mode.

## Getting started with spike shapes and templates

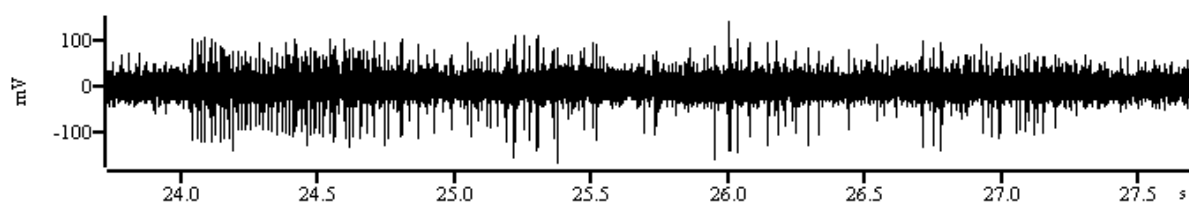
Sorting spikes is as much an art as a science; there is no substitute for the skill and experience of the investigator. Spike2 provides you with a toolbox of routines that will help you to discriminate various shapes, but there is no guarantee that two spikes are from the same source just because they have the same shape (or because any other measured parameter or set of parameters are the same).

### Quality of the original signal

Probably the most important contributor to successful spike sorting is the quality of the original data. If you can improve the signal to noise ratio, or adjust the electrode position to increase the amplitude of your target spikes relative to the background, the time spent will be rewarded by more certain classifications in less time.

The waveform sample rate used is also very important. For most spikes lasting 1 to 2 ms, a sample rate of 20 to 25 kHz is about right (some workers like higher rates than this). It is important that the data is frequency band limited to half the sample rate before it is sampled. For example, when sampling at 25 kHz there must be no frequency components above 12.5 kHz. If such components are present, they are aliased to lower frequencies and appear as noise that is very difficult to remove.

To check the quality of your signal it is a good idea to sample a section of data as a waveform first. Here is an example waveform sampled at 25 kHz from a tape recording.



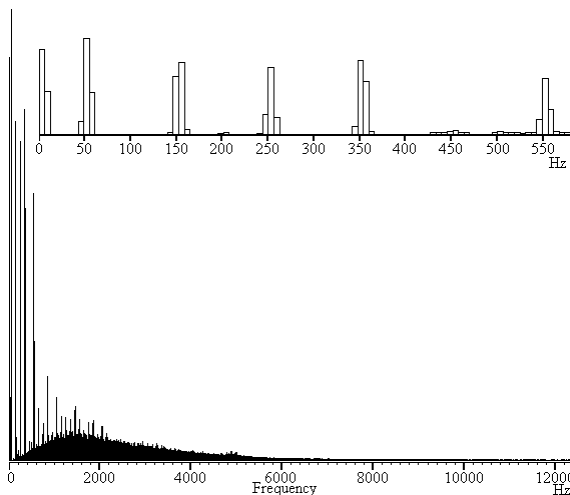
## Sampling configuration for this data

Open the sampling configuration Channels tab and click Reset. Click New Channel and select a Waveform channel. Set Channel to 1 and 1401 Port to 0. Set the Ideal adc sampling rate to 25000, and set Units mV = Input in Volts x 1000 + 0 and click OK. Now click the Resolution tab and set Optimise to Full. Finally click the Mode tab and select Continuous. You can now sample.

The example waveform amplitude is around  $\pm 100$  mV, which is a bit small. The 1401 family usually expect a full-scale input range of  $\pm 5$  Volts, and our spikes are using only 1/50 th of the available range. The size of each digitisation step is about 0.15 mV for the Power1401 or Micro1401. Although the system will work fine with signals of this amplitude, if your rig has more gain available, it is much better to make the signals around  $\pm 1000$  mV in amplitude. This gives you headroom for larger spikes and a good signal resolution.

Check the input for unexpected frequencies with the Analysis menu: New Result View: Power Spectrum command with a block size of 4096. Here we found narrow peaks at low frequencies and almost nothing above 6 kHz. Apart from a peak at 0 Hz caused by a small DC offset, the remaining peaks are all at odd multiples of 50 Hz. This data was recorded in England where the mains frequency is 50 Hz. Anything that reduces mains pickup (better shielding, removal of earth loops, cleaning up of connections) will make spike sorting easier.

Sharp power spectrum peaks often indicate signal contamination. If the peaks change frequency when you change the sample rate, they are caused by aliasing of frequencies above half the sampling rate and they may be removable with an external low-pass filter.



## Setting up the WaveMark channel

Assuming that we have adjusted the rig to get the best signal to noise ratio and spike amplitude, we must now set up a suitable sampling configuration to capture our spikes. We could sample the data as a waveform, but unless you really need all the data in between your spikes, this approach is wasteful of disk space. For example, a single waveform channel sampled at 25 kHz consumes 50 kB of disk space per second, or 3 MB per minute, or 180 MB per hour.

An alternative is to sample WaveMark data. In this case, each time the input signal crosses a positive or negative threshold we search for a peak or trough in the data and save the time and a small fragment of waveform. If we save 32 data points per spike, each spike costs us 80 bytes of data. As long as you set the threshold levels so that you get less than 625 spikes per second, this method uses less storage space. At a mean spike rate of 20 spikes per second, you would consume 96 kB per minute, or 5.8 MB per hour.

There is one further advantage to WaveMark data. As the data is captured, Spike2 can match it to templates and code each spike accordingly. This gives you further on-line analysis capability as you can look at the responses of individual spike types on-line. The on-line sorting is not the final word as you can review and resort the data off-line.

## Suggested configuration for WaveMark data

Open the sampling configuration Channels tab and click the Reset button. Then click New Channel and select a WaveMark channel. Channel should be 1 and 1401 Port should be 0. Set the Maximum sustained event rate to 30, and set Units  $mV = \text{Input in Volts} \times 1000 + 0$ . Set Points to 32 and Pre-trigger to 10 and WaveMark sample rate to 25000 and click OK. Now click the Resolution tab and set Optimise to Full. Finally click the Mode table and select Continuous. You can now sample.



Spike2 knows that you have a WaveMark channel in the sampling configuration, so when you click the Sample now! Button in the toolbar, it opens up a window for you to set trigger levels and adjust the number of sample points and to build templates.

When you open a window for the first time you are likely to have a more or less flat line across the centre of the display area and the trigger levels will not be in useful positions.

There are a lot of buttons and controls in this window. If you move the mouse pointer over one and leave it for a second or so, a line of text will “pop-up” with a short explanation.

### Home key or

This button is one of the most useful when you are setting up for spike shape capture. Each time you click the button, Spike2 will make a guess at reasonable trigger levels (and limit levels for the New WaveMark command with limits enabled) and adjust the display gain so that you can see the data. Once a trigger level is crossed a second trigger cannot occur until the waveform falls below half the trigger level.

It is likely that you will only wish to trigger in one direction, so move the cursor that you do not want to use to the edge of the window. To change a level, move the mouse pointer over the level, hold down the mouse button and drag to the new position.

It is possible to use both trigger levels, but we strongly suggest that you do not to avoid the sideways shift you will get if a spike triggers on the opposite side from the one you intended. If your spikes are biphasic, that is they have a rising peak and a falling peak, it is best to trigger in the direction that has the clearer peak. If your spikes are tri-phasic, as in the example above, it is best to trigger in the direction that has only one peak to avoid problems caused by false triggers on the second peak. If both directions are equally clear, choose the direction with the narrower peak (usually the first one).

### Up and Down keys or

Once you have got more or less the desired display gain you can use these two buttons to change the display scaling manually. You may also want to reduce the display gain so you can drag one of the trigger levels off the screen to reduce visual clutter. As an alternative to using this button, move the mouse pointer over the y axis and click and drag to scale the data.

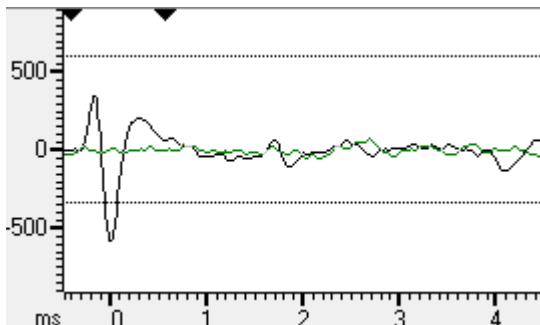
### Left and Right keys or


These buttons change the number of pre-trigger points. The trigger point is the first signal peak or trough after it crosses the trigger level, and is labelled 0 on the time axis. You can also click and drag the tick marks of the x axis to adjust the pre-trigger. You cannot change the number of pre-trigger points after you start to sample.


### PgUp and PgDn keys

These buttons increase and decrease the data points that are displayed and written to disk for each spike. The number of pre-trigger points is preserved if this is possible, so you should set the pre-trigger points first. You can scale the x axis around the zero point by clicking and dragging the x axis numbers.

You need to capture sufficient points to enable accurate spike identification. You also want to capture as few points as possible to minimise the data storage. In this example we have captured about 5 ms of data, which is at least 3 ms more than we need. As well as leading to huge data files, sampling too many points also increases the chance of the data for one spike containing data for a second or even a third spike. Further, Spike2 will not trigger for the next spike until it has finished sampling the previous one. You cannot change the number of points once you start sampling.



 The two black triangles mark the start and end of the region of the spike that is passed to the template matching system. Click on a triangle and drag to change the region. When you drag the markers, vertical cursors drop down to identify the selected area.

 The play control buttons from left to right are: step back, run back, pause, run forwards and step forwards. Step and run backwards are disabled during setup. There are keyboard short cuts for these buttons: **b** or **B** steps backwards, **N** runs backwards, **v** or **V** pauses, **n** runs forwards and **m** or **M** steps forward.

## Forming templates

By now you should be able to set up the system to capture spike shapes. The next step is to form templates from these spikes. The first step is to check that the template parameters are set to sensible values.

### Ctrl+Enter or

Click this button to open the template parameters. To get started, set these values:

### Suggested template parameters

In the New template section, set Number of similar spikes for a new template to 8, New template width

as a percentage of amplitude to 30, No template for spikes rarer than 1 in to 50. In the Matching a spike to a template section, set Maximum percent amplitude change for a match to 0, Minimum percentage of points in template to 60 and check the box for Use minimum percent only when building templates. In the Template maintenance section, set Template modification mode to Add All. In the Waveform data section, set the Waveform interpolation method to Linear, High-pass filter time constant to 20 ms and leave the Remove DC offset before template match unchecked. Finally click OK.

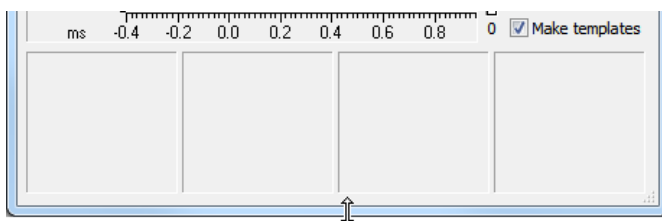
As you get more familiar with template matching you will probably change these values. However, this set is a useful starting position. We have turned off the amplitude scaling as this causes a large increase in on-line computation and so should not be used unless you really need it. Amplitude scaling comes into its own when your spikes change amplitude while preserving shape, for example during a burst.

### Del key or

Spike templates are displayed below the data display area. If you cannot see the template area drag the bottom edge of the window downwards with the mouse. The template area may already contain templates. If it does, you can clear them all by clicking the bin button at the top right of the window. You can remove individual templates by clicking the small bin button in the template window.

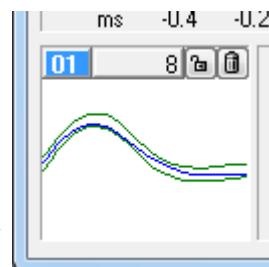
## Expanding the window to show the template area

The template windows have three sizes: small (in the picture), medium and large. You change from small to large by double clicking in the template area. Spike2 remembers the last window size you used for small, medium and large template displays. You can also expand the window sideways by dragging the right-hand edge of the window, or you can grab one of the corners to expand in both directions. As a short-cut, you can maximise and minimise the size of the window by double-clicking in the title bar.



To build templates, make sure that the Make templates box is checked, the play control is set to run forwards and that spikes are crossing the trigger level you set in the data display area. If all is well, when 8 similar spikes have been detected (8 is the number you set in the template parameters) a new template should appear in the template area.

In small mode, the four rectangles at the top of the template hold, from left to right, the code associated with the template (01), the number of spikes in the template (8), a button that indicates the locked state of the template and a button that can be used to delete the template. The vertical size of the displayed template is set by the vertical scale of the data display area.



As spikes are processed, the vertical bar on the right of the data display area indicates the number of templates that are being considered in grey and the number of confirmed templates in black. The first 20 confirmed templates appear in the template area.

The horizontal size of the template is set by the two black triangles in the data display area. For good spike sorting, it is important to select the region of the spike that shows the most variation between different classes of spikes. Do not include baseline areas before or after the spike. Spikes are matched to templates based on the errors between the mean template and each spike. Spikes are excluded from templates based on individual points falling outside the template boundaries.



You can control the appearance of the templates with these three latching buttons. If you click them once they become active, click them again and they become inactive. The leftmost button controls overdrawing of spikes in the template. With the button down, all spikes that match the template are over-drawn in the template. With the button up, only the last matching spike is drawn.

The centre button chooses between a display of the template boundary with the button down, and a display of the mean template with the button up. With the right-hand button up, spikes are displayed only in the template that they match. With the button down, spikes are also displayed (in grey) in all non-matching templates.



## Spike code and template colour

Each template has a code in the range 1 to 255. When the templates are used for spike sorting, the code is used to label matching spikes. Code 0 is not used by templates, but spikes can be given a zero code, meaning that they are not coded.

The code is represented by two hexadecimal digits; 1 is represented by 01, 2 by 02 and so on up to 9 which is 09. In hexadecimal (base 16 numbers), the numbers ten to fifteen are represented by the letters A to F, so ten decimal is 0A, eleven is 0B through to fifteen which is 0F. Sixteen decimal is 10 in hexadecimal, seventeen is 11 and so on.

We also allow the use of single character codes in place of the hexadecimal codes 20 to 7E. The keyboard marker channel uses the same coding system and the ASCII codes for the Roman printing characters lie in the range hexadecimal 20 to 7E. Although Spike2 can manage up to a hundred templates internally, it displays only the first twenty, so codes above 14 (decimal 20) are rarely used and single character codes are rarely a problem.

The code given to a template also determines the template drawing colour. In the View menu Change Colours... dialog you can assign colours for template codes 01 to 08 (or to a much higher code, if you choose), see the dialog for details. It is a good idea to set colours that contrast strongly with the background colour you have set for the time view as this colour also sets the background colour for the template windows.

New templates take the lowest available code, but you can change the template code by double clicking on it. This opens a dialog box in which you can edit the code. There is no restriction on the code you give, other than it must be in the range 01 to FF. In particular, you can have more than one template with the same code.

There are useful commands in the Templates menu. **Renumber** changes template codes to remove gaps caused by deleting or editing codes. **Order by Code** sorts the templates into ascending code order.

## Template operations

### Automatic template creation

With the play control set to run and the **Make templates** box checked, Spike2 creates templates automatically by looking for similar spikes and following the rules set in the template parameters dialog. Each new spike is compared first with the existing confirmed templates (displayed in the template area). If it doesn't match any of them it is compared with the provisional templates. If it matches nothing, it forms a new provisional template.

When a spike matches a template it modifies the mean template shape and width unless the template is locked. If the spike matches more than one template it is added to the one with the smallest error between the spike and the mean template shape.

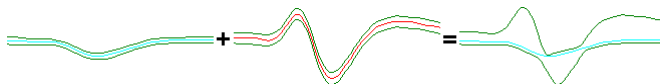
When a provisional template gets a new spike and this causes the template spike count to reach the **Number of similar spikes for a new template** set in the template parameters, it is ready to become a confirmed template. If it matches a confirmed templates it is merged with the confirmed template. Otherwise, a new confirmed template is created.

To avoid the system being swamped with provisional templates, there is a mechanism that makes them decay away. This is set by the **No template for spikes rarer than 1** in field of the template parameters dialog. If this number of spikes are tested and a provisional template does not get one new spike added to it, the template spike count is reduced by one. If the spike count becomes 0, the provisional template is deleted.

### Manual template creation and merging

Instead of letting the system create templates, you can build them yourself. Use the play control to step forwards one spike at a time until you find one that you want to turn into a template. Click the mouse in the middle of the display area and drag the spike to an unused window in the template area and release. If you drag a spike shape to an existing template, the spike is added to it.

You can combine two templates by dragging one of the templates and dropping it on the other. Spike2 will align the dragged template with the target and then merge the two shapes by combining the template limit.





---

This example shows the effect of merging two very dissimilar templates. You can see that the extremes of the template boundaries of the original templates are preserved in the result. Normally, you would merge two similar templates. You can test for template similarity by dragging a template over another without releasing.

### Number of templates

You can create (and Spike2 will remember) up to 20 templates per channel. However, if you are preparing for on-line sampling, only the first 8 templates in the template area will be passed to the 1401 for use on-line. You can, of course, resort the data off-line using all 20 templates, and it is possible to sort spikes into many more classes than this by using the Analysis menu Marker Filter and Edit WaveMark commands.

### Locking templates

Each time a spike matches a template, and the Make templates box is checked, the spike is added into the template unless the template is locked. The lock button in each template can be used to lock and unlock templates manually. You can also make the templates lock automatically after a set number of spikes have been accumulated by setting the Template modification mode in the template parameters dialog to AutoFix.

If you don't lock the templates, they change as more spikes are added. This means that the shape you end up with could be quite different from that with which you started. For example, if you create a template manually from a specific spike, you may want it to keep the exact shape so you would lock it.

## Amplitude variation

If you find that the system keeps generating multiple templates for the same spike because the spike amplitude changes, there are several things to try. The obvious solution is to set the Maximum percent amplitude change for a match to a non-zero value. For example, if you set this to 30%, each spike can be increased or decreased in size by up to 30% before attempting to match it to the templates. Spike2 does this by computing the area between the spike and the zero baseline and changing the spike amplitude so that this area is the same as the area between the template and the baseline.

Beware that this is computationally expensive, which does not matter for off-line sorting, but it can be important on-line, especially if you don't have a Power1401. The more channels of spikes and the more templates you are matching against, the larger the time penalties you will impose. If you find that turning this option on leads to spikes not being classified when the spike rate is high, then you may need to consider other options.

Another method to cope with amplitude variation is to make the templates wider. You can do this by merging templates, or by increasing the New template width as a percentage of amplitude in the template parameters. However, wider templates leads to less discrimination between spike shapes.

Closely allied to making the templates wider is to reduce the Minimum percentage of points in template field. However, this also leads to less discrimination between spike shapes.

You might also consider checking the Use minimum percent only when building templates box. If you do this, when you run on-line and when reclassifying spikes, spikes are matched to the template that has the smallest error between the shape of the spike and the shape of the template, the template limits are ignored. This option is most useful when you know that a whole bunch of spikes are either type A or B but the data is very noisy, so you chose a good spike of each class to make two templates, then you check the box and sort on the basis of minimum error.

If you do not need to separate many shapes on-line, you can accept that it takes 2 or even 3 templates to cover the full range of amplitudes and give the 2 or 3 templates the same code (double click the code and edit it).

## Using sound

If your computer has a sound card, you can play a sound for each spike. Spikes that match a template play a sound with a pitch that rises as the code for the template increases. Spikes that don't match will play a lower pitch. How successful this is depends on the capabilities of your sound system! You may find that listening to the raw signal wired to the input of an audio amplifier is a better solution.

## Sampling data

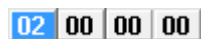
Although we have not covered every possible control you could have used, you should now have enough information to set up your system with some templates and sample data. Click the **Sample Data** button and the dialog box will close and a new data file window opens. Closing the template setup dialog, fixes the number of points that will be captured for each spike and the number of pre-trigger points. If you followed our sampling configuration settings, the new data window will contain a channel of WaveMark data and the Keyboard marker channel.

## Monitoring spikes

You can adjust the trigger levels and even change the templates during sampling with the **Analysis** menu **Edit WaveMark** command. If you use this a window that is very similar to the template setup window opens.

The display area shows the last spike that crossed a trigger level, and when there are no spikes, it shows the latest data. You can adjust the trigger levels by dragging them, and *the level change is passed to the 1401 immediately*. If you make any other change, this has no effect on the 1401 unless you click **Online update**.

The 1401 holds a set of templates that it uses to classify spikes. There is a separate set held by Spike2 that start out as identical to the ones in the 1401; you can change them by adding spikes, deleting templates or creating new ones. Click **Online update** to copy the latest Spike2 templates to the 1401.



The leftmost of these 4 buttons shows the marker code assigned by the 1401; the other three buttons are not used on-line and are disabled. The **Best match** field displays the marker code of the template in the template area that best matches the spike in the display area. These will usually be the same. They can differ if you have edited the templates, or if **Make templates** is checked so that the template area is changing.



If you enable sound, the tones you hear depend on how the spikes match templates in the template area; the tones do not depend on the classification done by the 1401. This is to allow you to modify templates in preparation for **Online update**.



This is a latching button that is normally down for on-line use. Click the button to change the state between down and up. With this button down, the displayed spikes are always taken from the end of the data file, that is from the spikes that have just been sampled. If the spike rate is too high to show all the spikes, spikes are skipped.

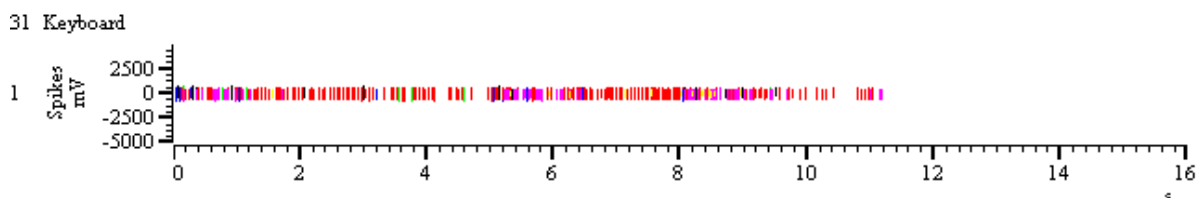
With the button up, a new cursor appears in the time view to mark the point from which spikes are taken, all the play control buttons become enabled and the windows behaves almost identically to the off-line **Edit WaveMark** window which is described later. The marker code buttons still only allow the first marker code to be used; you can use all four codes when off-line.

## Monitoring multiple channels

If you have multiple channels of WaveMark data, you can monitor them all by opening the Spike Monitor window (use the **View** menu **Spike Monitor** command). This view allows you to check all the WaveMark channels at a glance. This window also has links to the **Edit WaveMark** dialog. If you click on a channel in the Spike Monitor window, this selects that channel in the **Edit WaveMark** dialog, ready for adjustment.

## Drawing modes for spikes

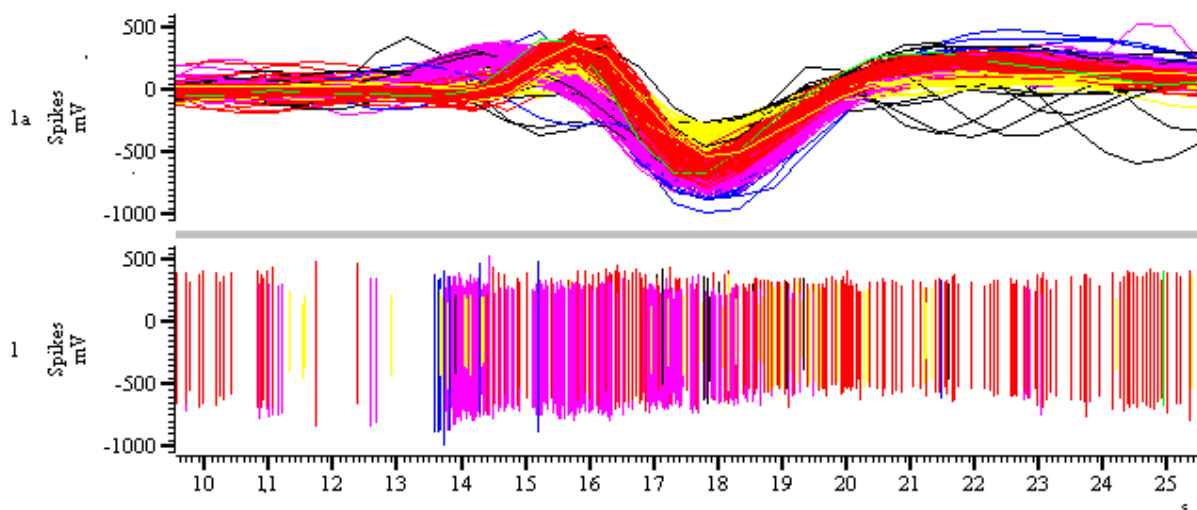
If you use the **View** menu **Channel Draw Mode...** command and select the WaveMark channel, you will find that you have a very wide choice of display modes. To start with, display the channel as **Waveform**. Arrange the x axis to show 10 to 20 seconds worth of data and click the **Start** button in the sampling control bar or use the **Start Sampling** command in the **Sample** menu. You should see something like the following:



If no spikes are appear, use the Analysis menu Marker Filter... command and select each of the four layer in turn and click All, then click OK. We will discuss this command in more detail later.

The spikes are colour coded for the template that they match, or are drawn in black if they don't match any template. These spikes are rather small, so the first thing to do is to double click the y axis and optimise the display. You can display the codes for the spikes as well as the waveforms by changing the drawing mode to WaveMark, but this is not very useful when you have a lot of spikes in the window as the codes will overwrite each other. Drawing in WaveMark mode on-line is much slower than drawing as Waveform, so we do not recommend this mode for on-line use.

Another useful mode is Overdraw WM (overdraw WaveMark). This gives an immediate overview of the spike sorting. To see this mode in action, use the Analysis menu Duplicate Channels command to duplicate the WaveMark channel. Then use the View menu Channel Draw Mode... command to set the drawing mode for the duplicated channel to Overdraw WM. You may need to adjust the size of the data window to generate a useful image. In the next example we have hidden the keyboard channel.

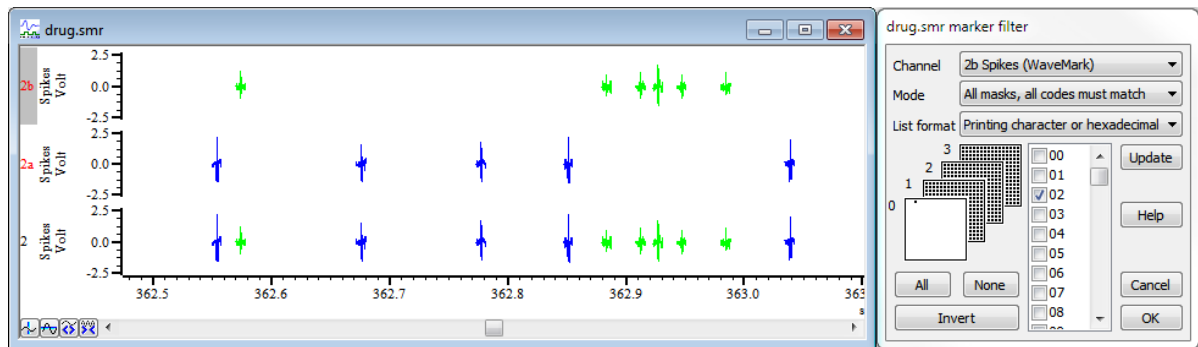


In Overdraw WM mode, the full width of the window is used for each spike. There is a grey bar between the overdraw area and the rest of the window to show that the x axis does not apply. When the window scrolls, new spikes are added to the overdraw area, but old spikes are not removed as this would force the entire window to redraw, which could take a long time. If you want to force the overdraw area to redraw, click or drag the thumb in the scroll bar at the bottom of the window.

You can locate a particular spike in Overdraw WM mode. Move the mouse pointer so that it is over the desired waveform at a place that is clear of all other waveforms, right-click and select Find with cursor 0 from the pop-up context menu. This moves cursor 0 to the position of the event. If the Edit WaveMark dialog is open, it will also move to the position as cursor 0 controls where the dialog collects data.

## Using the Marker Filter

Although it is nice to see all the spikes you have collected, for most purposes, you are interested in the behaviour of one unit, or the relationship between one unit and another. If your templating and spike sorting has gone well, you will have isolated each unit as spikes with a particular code (or possibly more than one code). You now need a way to treat each unit as a separate channel of data.



### The hard way

In the example shown, channel 2 has two classes of spike, sorted into codes 1 and 2. You could generate channels 2a and 2b by duplicating channel 2 twice (right click on the channel and select Duplicate from the context menu). Then right click on channel 2a and select the Marker Filter command, clear the check box next to 01 and click the Invert button. This clears all check boxes except the one next to 01. Then click Update and channel 2a will display only spikes with code 01. Next, select channel 2b in the channel list and repeated the procedure for code 02.

### The easy way

However, there is a much easier way to do this. The Edit WaveMark dialog has a button that generates a duplicate channel with the marker filter set to display each template code. Alternatively, you can use the Edit WaveMark dialog Analysis menu Duplicate command (short-cut `Ctrl+D`).

The result of this is that we have two new channels, each with spikes from a single unit. If your sorting resulted in one unit having more than one code, then you should select all the codes for that unit in the Marker Filter dialog; there is no need to sort spikes so that one template matches every single spike from the unit.

If you decide that spikes with several different codes should all have the same code, you can use the new Set Marker Codes command in the Analysis menu, or right click on a channel and select this command from the context menu. Set the marker filter for the channel to display the codes that you wish to amalgamate then use the Set Marker Codes dialog to change all the spikes with these codes to the same code.

## Creating on line templates with cluster analysis

Cluster analysis is normally used off-line. This is because extracting principal components and automatic clustering can take a noticeable time if you are working with a large number of events; also the clustering analyses are not available from the on-line template setup dialog. However, you can use clustering to create on-line templates. There are two ways to do this:

### Off-line clustering

Sample as normal and set the spike trigger levels in the on-line template setup dialog. There is no need to create templates at this stage. Start sampling and capture enough data to be able to form templates, and then stop sampling. Open the Edit WaveMark dialog and from the Analyse menu use the clustering method of your choice to classify the captured data.

When you are satisfied with the clusters, use the cluster dialog File menu Apply command to classify the original data. This will also clear all templates for the channel from the Edit WaveMark dialog and build new templates based on your classification. Now close the Edit WaveMark dialog. This will save the templates in the sampling configuration as off-line templates.

Now sample again, and when the on-line template setup dialog opens, use the File menu Load and Save command (short-cut `Ctrl+S`) and select the off-line template for your channel. This will load the templates that you created off-line.

### On-line clustering

We do allow you to use the Edit WaveMark dialog on-line to monitor spikes and you can also use the clustering commands on-line. However, the clustering functions will run more slowly than when working off-

line; having more than one processor in your computer will help here. If you have large data files and huge numbers of events or a high data throughput to disk, this may not be a viable procedure as the system may not be responsive enough to be useful.

If you are using timed or triggered sampling, data sections that are not marked for writing to disk are volatile; data that the clustering code expects to find may have vanished by the time you try to apply changes.

When you apply the results of clustering on-line, the newly created templates are copied to the 1401 as if you had used the **Online update** button.



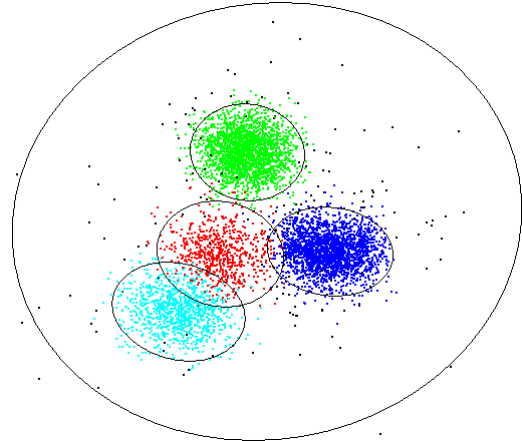
# **17: Clustering**

# Clustering

## Introduction

Clustering is the generic term for methods that group similar objects into classes. In our case, the objects are spikes that consist of 1, 2 or 4 waveform traces of typically 20 to 40 data points each. The number of objects to cluster will often be large; clustering tens of thousands of spikes is not at all uncommon, so the algorithms used must be fast enough to manage large data sets in a reasonable time. Spikes also form a temporal sequence and we have to take into account that the clustering properties may change with time.

If you have  $n$  data values that define each object, you can cluster in  $n$  dimensions. However, if  $n$  is greater than 3 it is very difficult to visualise the clusters. For spikes,  $n$  is typically 30 or more, so we need ways to extract 2 or 3 independent measurements from our data that maximise the differences we care about between the classes and minimise the differences due to noise in the data. Spike2 provides four methods:



- |                              |                                                                                                                                                                                                                                                                                                                              |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Principal component analysis | Principal Component Analysis (PCA) is a mathematical procedure that automatically extracts the features from your data that contribute the most to the differences between the waveforms that make up your spikes. This is usually the method to try first as it is less susceptible to noise than feature measurements.     |
| Feature measurements         | Feature measurements are user-defined values extracted from each spike, such as amplitudes, latencies, areas and slopes. Each trace is parameterized based on positions of peaks and zero-crossings. You then choose two or three measurements that emphasise the differences between the spike classes to cluster the data. |
| Template correlation         | You select two or three spike templates, then the $x$ , $y$ and optionally $z$ parameters of the clustering are the waveform correlations of each spike with the selected templates. The correlations are amplitude independent, so this can be useful with spikes that maintain shape but change amplitude.                 |
| Template errors              | You select two or three spike templates, then the $x$ , $y$ and optionally $z$ parameters of the clustering are the sums of the squares of the errors between each spike and the three selected templates.                                                                                                                   |

## Extracting clustering values

All methods generate a table of values with one row per spike. Each table row holds the spike time, a class code and the  $x$ ,  $y$  and  $z$  values derived from the spikes. All clustering operations operate on the data in this table, not on the original spikes in the data file; you can choose to apply classification changes to the original data at any time. We call the items in the cluster window *events* to distinguish them from the original spike data.

The clustering dialog contains tools that make it easy to visualise the clusters in two and three dimensions and tools that allow you to classify the clusters either manually or with various degrees of automation. How well automatic clustering works will depend on how well separated the clusters are.

## Using cluster analyses

The starting point for cluster analysis is the Edit WaveMark dialog. The clustering commands are in the dialog's Analysis menu. The analyses operate on data in the time range set in the Edit WaveMark dialog and honour marker filters set for the data channel. Changing the Edit WaveMark channel closes any associated clustering windows.

The clustering dialog is linked to the Edit WaveMark dialog. The current event in this dialog flashes in the



cluster window. If you click on an event in the cluster window, the Edit WaveMark dialog will jump to the spike that generated the event; when you apply the results of the cluster analysis, the Edit WaveMark dialog will re-evaluate its templates to match the new classifications.

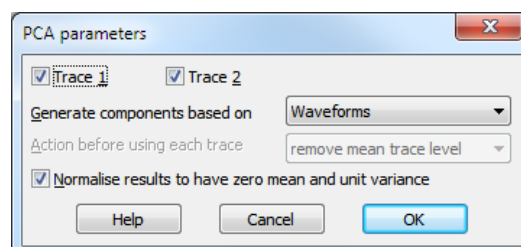
### Reasonable limits

Although we try very hard to make drawing the clustering images as fast as possible, with the hardware commonly available in 2013, once you have more than a few hundred thousand spikes it takes an appreciable time to recalculate the display, so the animation of the clustering becomes jerky. Further, the PCA analysis takes a time proportional to  $m*n^2$  where  $m$  is the number of spikes and  $n$  is the number of data points that represent the spike. You may find that you want to limit the value of  $m*n^2$  to around 1 billion to avoid very long waits for the PCA analysis to complete.

## Principal Component Analysis

To use Principal Component analysis, open the Edit WaveMark dialog (make sure you are not in collision analysis mode) and drag the black triangles at the top of the data display area to select the region of each spike to process. Ideally you should exclude baseline regions from the spikes to reduce noise and improve the cluster separation. The time range in the data file to process is set by the **Set Time Range** command in the dialog Control menu.

The Principal Components command in the Edit WaveMark dialog Analysis menu opens the PCA parameters dialog. If your spikes have a single trace, everything except the Normalise measurements check box is disabled. With more than 1 trace (stereotrode and tetrode data), all dialog items can be used. The items are:



### Trace n

When you have more than one trace you can choose which to use for analysis. If you clear all the check boxes, Trace 1 is selected automatically. Some of the other options in the dialog require at least 2 selected traces; reducing the number of selected traces may change other dialog fields.

### Generate components based on

This field allows you to choose the data that the principal components are generated from when you have more than one data trace. You can choose from:

|                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Waveforms                 | The data source is the waveform region of each spike set in the Edit WaveMark dialog.                                                                                                                                                                                                                                                                                                                                                                                   |
| Amplitude at time 0       | The data source is the peak amplitudes of the selected traces.                                                                                                                                                                                                                                                                                                                                                                                                          |
| Mean amplitude and ratios | The data source is the mean peak amplitudes (ignoring sign) of the selected traces and the ratios of the peak amplitudes to the mean peak amplitude.<br><br>The peak amplitude is found by searching each trace for the peak (or trough) that is nearest to the trigger point. The trigger point is at 0.0 milliseconds in the Edit WaveMark data display window. Traces are pre-processed as set by the Action before using each trace field before locating the peak. |

If there is not enough data to generate three output values, a z value is not generated and you will not be able to rotate the data around the x or y axis. This will happen if you choose a mode other than Waveforms and have 2 selected traces. It can also happen in Waveforms mode if the selected spike region contains less than three data points.

### Action before using each trace

This field allows you to decide what pre-processing should be applied to each trace before it is used. In Waveforms mode the mean level of the selected region of each trace is always subtracted. In other modes you can choose from:

- |                         |                                                                                                                                                                                                                                              |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| No trace pre-processing | This trace is used exactly as read from the data file.                                                                                                                                                                                       |
| Remove mean trace level | The mean level of the trace is subtracted before using the data. You can use this to remove baseline drift.                                                                                                                                  |
| Subtract best-fit line  | A straight line is fitted to the data of each trace and subtracted from the trace before the data is used. If your spikes suffer from sloping baselines you can use this method to remove them. This also removes DC offsets from your data. |

### **Normalise measurements**

If you check this box, the x, y and z components generated by the principal component analysis are shifted and scaled so that they all have a mean of 0 and a variance of 1. This can make the data easier to manipulate in the clustering dialog. If you do not check this box, the displayed axes in the cluster plot originate at (0, 0, 0); events close to this point are small spikes, and are more likely to be due to noise.

### **Performing the analysis**

Click the OK button to analyse the data. This can take a long time if you have a many tens of thousands of spikes and many data points per spike. If the analysis takes longer than one second, a progress dialog appears with a Cancel button that allows you to abandon the analysis and return to the Edit WaveMark dialog. If a marker filter is set for the channel, only spikes that match the filter are processed.

## What is Principal Component Analysis

Principal component analysis reduces a large set of non-independent data into a set of orthogonal components that is ordered in terms of the significance of each component to the whole dataset. Orthogonal means that if you multiply and sum corresponding elements of two components, the result is zero. In terms of the Spike2 script language, `ArrDot(Ci, Cj)` where  $C_i$  and  $C_j$  are two components is 0 for  $i < j$  and 1 for  $i = j$ . For most sets of spikes, only the first few components are useful, the rest correspond to noise. For clustering, the x, y and z values are the proportions of three user-selected principal components in each spike (usually the first three).

There is no need to be familiar with the mathematics behind principal components to make effective use of it. However, some knowledge of how it works may help you to understand its limitations and let you decide it is appropriate for your data. Principal Component Analysis is based on Singular Value Decomposition (SVD), which can be summed up by the matrix equation:

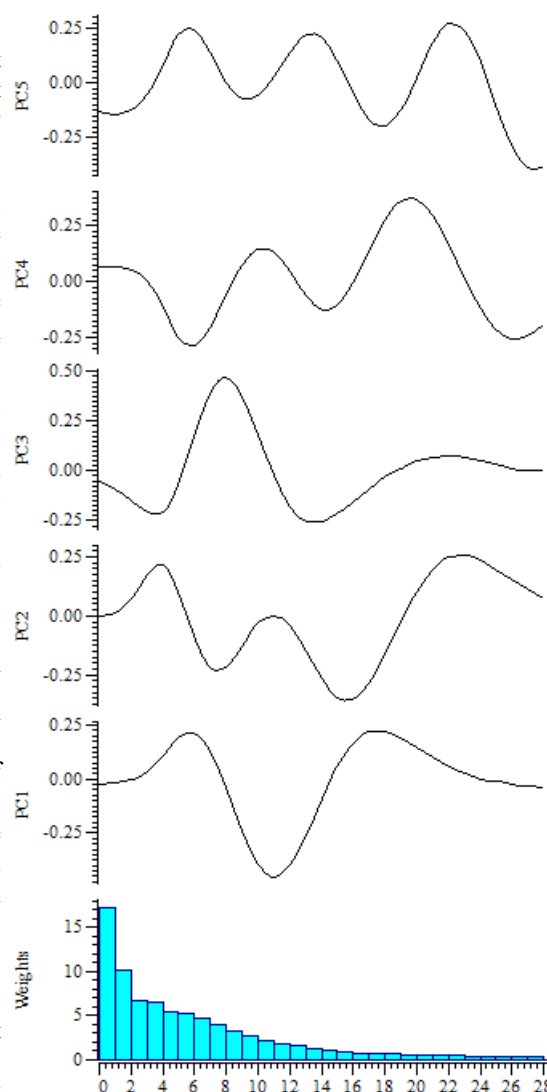
$$\mathbf{X} = \mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}'$$

Where  $\mathbf{X}$  and  $\mathbf{U}$  are matrices with  $m$  rows by  $n$  columns,  $\mathbf{W}$  is an  $n$  by  $n$  diagonal matrix and  $\mathbf{V}'$  is the transpose of an  $n$  by  $n$  square orthogonal matrix. When applied to spike waveforms,  $m$  is the number of spikes,  $n$  is the number of data points in each spike. The rows of the matrix  $\mathbf{X}$  are the spike waveforms with their mean value removed. The matrix  $\mathbf{V}$  holds the principal components, the diagonal matrix  $\mathbf{W}$  holds the variance of the original data that each component accounts for and each row of  $\mathbf{U}$  holds the contribution of each component to the spike in the corresponding row in  $\mathbf{X}$ .

The principal components are ordered so that the first principal component contributes the most variance in the original data, the second represents the most remaining variance after removing the first component, and so on. If there are  $n$  points in the original spike waveform, you will get up to  $n$  principal components.

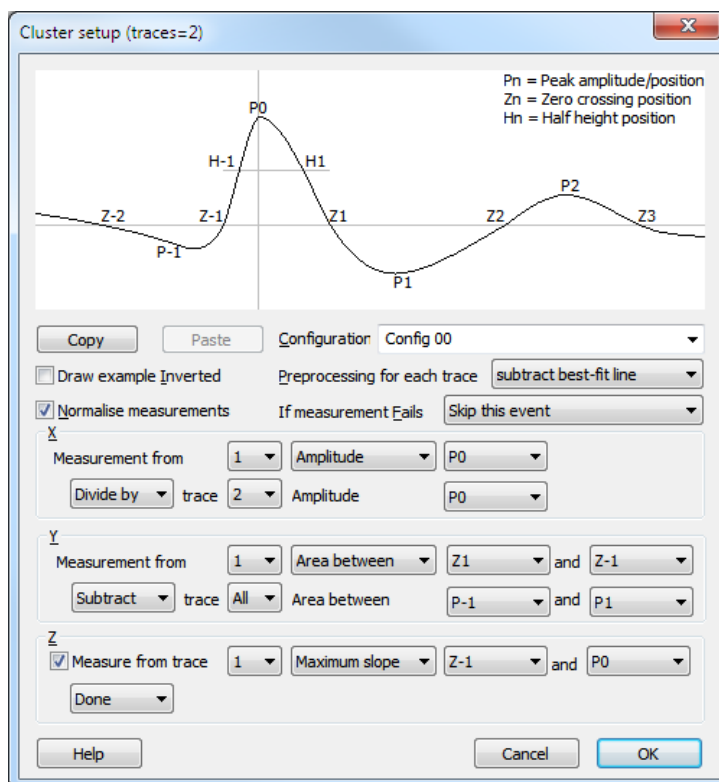
The time taken to compute  $\mathbf{U}$ ,  $\mathbf{W}$  and  $\mathbf{V}$  from  $\mathbf{X}$  is proportional to  $m \cdot n^2$ ; this means that doubling the number of spikes doubles the time, doubling the number of data points per spike quadruples the time. This is why you should not include baseline data for each spike; not only does it add noise to the analysis, it also takes longer.

If you want to see what the components look like, the File menu Principal Components option of the clustering dialog creates a result window holding the principal components and how much each of them contributes to the original data set.



## Cluster on measurements

When you select Cluster on Measurements from the Edit WaveMark dialog, a new dialog opens in which you can choose either 2 or 3 measurements to take from the spike data for clustering. The entire spike trace is used for measurements. The time range in the data file to process is set by the Set Time Range command in the Edit WaveMark dialog Control menu.



The dialog has 4 main regions: a schematic of a spike trace to use as a reference when selecting measurements, general controls, definitions of the x, y and z measurements, and the buttons to close the dialog. The dialog controls are:

### Configuration

There are 10 measurement configurations, selected with the Configuration combo box. Configurations for 1, 2 and 4 traces are stored separately. Initially, the configurations are labelled as Config 00 through Config 09, but you can edit these names to describe the type of measurement. You can copy a configuration to a different position in the table. Click the Copy button on the configuration you want to copy, select a different configuration and click Paste to overwrite it.

### Draw example inverted

The example waveform at the top of the diagram can be drawn either way up. This control lets you choose the direction that best suits your data.

### Preprocessing for each trace

If your data has a large DC offset or has a sloping baseline, measurements may not generate the expected results. You can choose to process each trace in your data before the measurements are made. You can choose from:

#### *No trace pre-processing*

This trace is used exactly as read from the data file.

#### *Remove mean trace level*

The mean level of the trace is subtracted before using it.

#### *Subtract best-fit line*

A straight line is fitted to each trace and subtracted from it before the data is used.

### If measurement fails

It is not always possible to make a measurement. For example, a peak might not exist in the data. You can choose between:

*Use standard work-around*

This sets un-measurable amplitudes to 0 and un-measurable times to a time just past the end of the spike when searching forwards or to a time just before the start of the spike when searching backwards.

*Skip this event*

The spike is not included in the measurements or clustering. If you use the **Apply** command in the cluster dialog, all skipped spikes are given code 00.

**Normalise measurements**

If you check this box, the measurements for x, y and z are shifted and scaled so that they all have a mean of 0.0 and a variance of 1. This can make the data much easier to work with in the cluster dialog, especially if you rotate the data. Of course, the measurements are then in arbitrary units, so you may not wish to do this if you want to export the values as text from the cluster dialog.

**X, Y and Z measurements**

These three areas are identical except that you can enable and disable Z measurements with the **Measure from trace** check box.

We model each trace by searching outwards from time 0 for the peak P0. We expect this peak to exist because the spike data was aligned on a positive or negative peak when it was captured. Next we search outwards in both directions from P0 to find the zero crossings Z-2, Z-1, Z1, Z2 and Z3. Then we find the peaks P-1, P1 and P2. Finally we locate the half peak height positions H-1 and H1. To make the searches as accurate as possible, we fit a cubic spline through the trace data points and search for peaks and level crossings using the fitted data. In addition to these measurements, the following were added at version 7.02:

**PkMax** The position and amplitude of the highest (most positive) peak.

**TrMin** The position and amplitude of the minimum (most negative) trough.

**Max** The position and amplitude of the most positive point in the trace.

**Min** The position and amplitude of the most negative point in the trace.

The top line of each area sets a basic measurement. The first field on the second line can be set to one of **Done** to use this value or **Subtract** or **Divide by** to make a second measurement and subtract this from the first or divide the first measurement by the second. You can choose from the following measurements:

|                       |                                                                                                                                                                                        |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Amplitude</b>      | You can select any of the peaks (P-1 to P2) as the measurement. The units of measurement are the y axis units of the channel.                                                          |
| <b>Time at</b>        | The measurement is the time of any feature (Peak or level crossing). The units are milliseconds relative to the peak trigger point (the 0 time in the Edit WaveMark dialog).           |
| <b>Area between</b>   | The measurement is the area between any two features, from the curve to the y axis zero. The units of the area are y axis units times milliseconds. All areas are treated as positive. |
| <b>Maximum slope</b>  | The measurement is the value of the steepest slope of the fitted cubic spline between any two features in y axis units per millisecond.                                                |
| <b>Best fit slope</b> | The measurement is the slope of the least-squares best-fit line between any two selected features in y axis units per millisecond.                                                     |

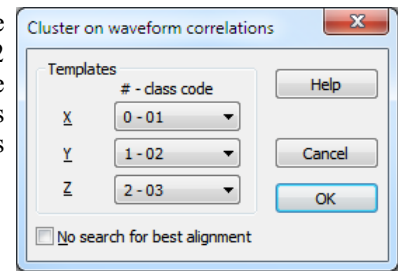
If you are working with multiple trace data you can choose which trace to use as a source of your measurement. If any part of the measurement calculation uses a failed feature measurement, the measurement is marked as failed. You can also choose **All traces**, in which case the measurement is the average measurement value on all of the traces.

**Running the analysis**

Click on the **OK** button to save any changed configuration settings and generate the measurements. The analysis process takes a time proportional to the number of spikes. Once the analysis is done the Cluster dialog will appear. The **Cancel** and **OK** buttons change to **Close** and **Apply** if you open this dialog with the **Reanalyse** command from the clustering window.

## Cluster on template correlations

To use this option, click on the Cluster on Correlations command in the Analysis menu of the Edit WaveMark dialog. There must be at least 2 templates defined in the dialog for this command to be enabled. The command opens a dialog in which you can choose two or three templates to correlate each spike with to generate the X, Y and optionally Z values for clustering. You can choose None as an option for the Z value.



### No search for best alignment

If you check this box, the section of the spike shape correlated against each template will be the section marked for template formation in the main display in the Edit WaveMark dialog. If you leave the box unchecked, Spike2 will search around the section marked for template formation for the best correlation. This search extends for up to two sample points in either direction and uses cubic spline interpolation of the spike waveform to estimate values between sample points.

You will normally leave the box unchecked unless you have chosen a template region that does not include the spike alignment point (usually the peak or trough), and the time delay of similar shapes is the distinguishing criterion. If you check the box, the analysis is much quicker, but the result is usually significantly worse.

### How the correlation is calculated

The correlation between a segment of the spike shape starting at position  $s$  and a template of  $n$  points is calculated as follows:

1. Make a spike segment of  $n$  points (use cubic spline interpolation if  $s$  is not integral).
2. Normalise the spike segment by shifting and scaling so that the mean value is zero and the sum of squares is unity.
3. Normalise the template to have zero mean and unity sum of squares.
4. The correlation is the sum of the point by point product of the two waveforms.

The result of this is a value between 1.0 (if the spike and template have the same shape) and -1 (if they have the same shape but one is inverted with respect to the other). If the two shapes are significantly different, the result will be nearer to 0 than to 1 or -1.

### When to use correlations

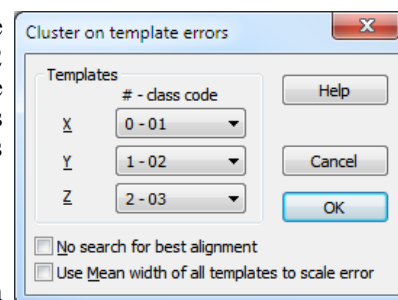
The correlation has the property that it is *independent of the amplitude of the spike*; the correlation is purely a function of shape. As spike amplitude is often the single most important feature when sorting, you should consider carefully whether this is worth losing. On the other hand, if you have a spike of more or less constant shape, but variable amplitude, this may be exactly what you want.

We anticipate that this method will be useful in sorting similar spikes with varying amplitudes into two or three categories, possibly after using other sorting methods to reduce the problem down to the difficult cases.

The results produced in three dimensions tend to be a thin layer of events with patches at a radius of 1 from the origin for each of the templates, which may not be ideal for the automatic clustering. You may find that you have to use manual event selection.

## Cluster on template errors

To use this option, click on the Cluster on Errors command in the Analysis menu of the Edit WaveMark dialog. There must be at least 2 templates defined in the dialog for this command to be enabled. The command opens a dialog in which you can choose two or three templates to compare each spike with to generate the X, Y and optionally Z values for clustering. You can choose **None** as an option for the Z value.



### No search for best alignment

If you check this box, the section of the spike shape compared with each template will be the section marked for template formation in the main display in the Edit WaveMark dialog. If you leave the box unchecked, Spike2 will search around the section marked for template formation for the smallest error. This search extends for up to two sample points in either direction and uses cubic spline interpolation of the spike waveform to estimate values between sample points.

You will normally leave the box unchecked unless you have chosen a template region that does not include the spike alignment point (usually the peak or trough), and the time delay of similar shapes is the distinguishing criterion. If you check the box, the analysis is much quicker, but the result is usually significantly worse.

### Use Mean width of all templates to scale error

With this unchecked, the error at each comparison point is scaled by the template width at that point. If you check the box, the error is scaled by the mean width of all templates (with multiple traces, the error for a trace is scaled by the mean template error for that trace in all templates). If you check this box, the error can be thought of as the least squares error. If you don't check it, the error is more like a chi-squared error based on the template width. Use whichever gives the better separation.

### How the error is calculated

The error between a segment of the spike shape starting at position  $s$  and a template of  $n$  points is calculated as follows:

1. Make a spike segment of  $n$  points (use cubic spline interpolation if  $s$  is not integral).
2. Adjust the spike segment to have a mean value of zero.
3. Adjust the template to have zero mean.
4. Form the sum of the squares of the point-by-point difference of the two waveforms divided by either the point-by-point width or by the mean width.
5. Divide the result by the number of points to form the average error per point.

The result of this is a value from 0 upwards. You would expect values for a matching template to be less than 1.

### When to use errors

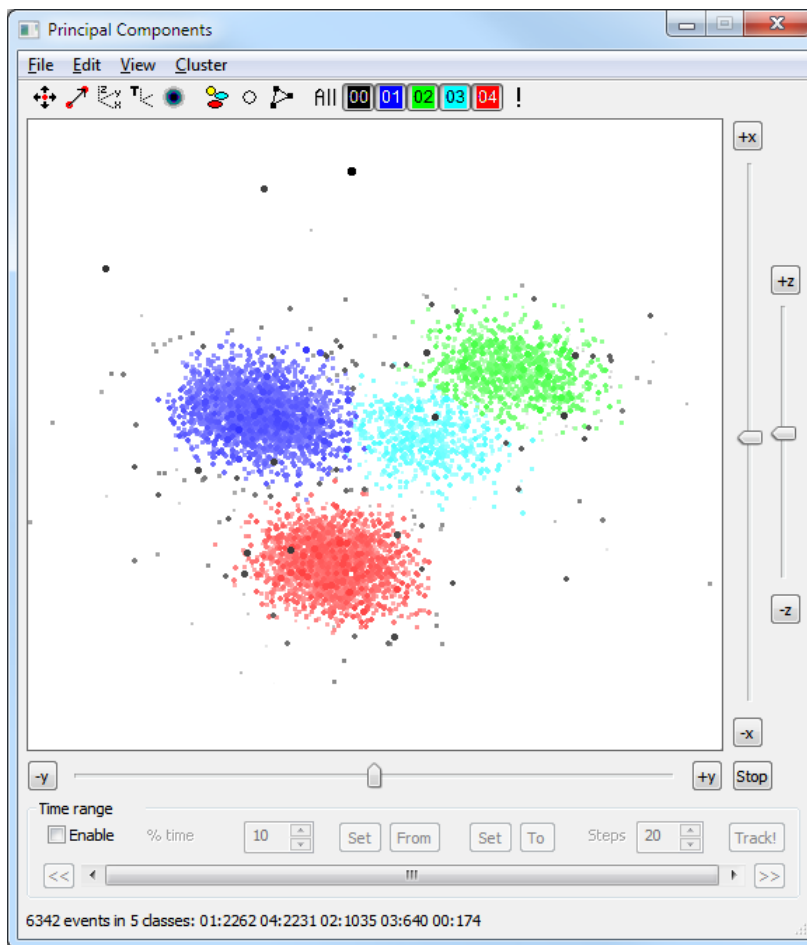
We anticipate that this method will be useful in sorting similar spikes into two or three categories, possibly after using other sorting methods to reduce the problem down to the difficult cases. Spikes that resemble each of the templates should lie in the range 0 to 1 on the axis that corresponds to the template.

If you check the *Mean width* box, this method is similar to the method used for template matching in the Edit WaveMark dialog and you can use it to get a visual indication of the reliability of the separation achieved by the template method. If this produces non-overlapped clusters, it is likely that the template method is reliable.

## The Clustering dialog

You reach this dialog by using one of the clustering commands in the Analysis menu of the Edit WaveMark dialog. The dialog behaves almost identically for all analysis methods; the dialog title indicates the source of the data. The dialog contains a menu, a toolbar that selects what to display, the cluster window, sliders for rotation around the x, y and z axes, the Time range area and a status line. You can hide the rotation slider

controls and the Time range area.



*Clustering dialog (click features for more information)*

You can resize the dialog by clicking on any edge or corner. The minimum size depends on the displayed items; if you hide the Time range area and the rotation controls you can make the dialog very small indeed!

The analysis you selected built a table of events. Each event contributes one dot to the display. For each event, the table holds the associated spike time, the x, y and z values from the analysis and the class code. The initial class codes in the table are copied from the spikes. Each spike has 4 marker codes; we use the marker code selected in the Edit WaveMark dialog. Class assignments made in this dialog only change the table. The File menu Apply changes command copies changes to the data file. There are two methods you can use to display the events, Density Plot and Class code colours.

### Density plot

Select this display mode with the View menu Density Plot command. The display shows the density of events in each pixel. The View menu Density Colour Map command lets you change the colour scale used to indicate the event density. The View menu Density Settings command gives you control over the creation of the density map.

### Class code colours

In this display mode, the colour of each dot is set by the class code (matching the WaveMark colours used in the time window) and the background colour for the cluster region is set in the main Spike2 colour palette.

### Current event

The event corresponding with the spike displayed in the Edit WaveMark dialog blinks on and off if it is in the visible area. This event blinks, even if it is not part of the currently selected set of event; this is for information only. The event blinks between the class colour of the event and the background colour. You can change the current event by clicking on a different event; this will change the current event here and in the Edit WaveMark dialog, and will scroll the associated time view if it is tracking the Edit WaveMark dialog. If you



have Minimum Intervals enabled, clicking near an arrow indicating a short interval will set the event that starts the interval as the current event.

## Class ellipsoids

We calculate statistics for each class of events and based on the assumption that the event  $x$ ,  $y$  and  $z$  values in each cluster are normally distributed about the cluster centre, we calculate ellipsoids of constant probability around the cluster centres. You can set the size of the ellipsoids in terms of the Mahalanobis distance (this is multi-dimensional version of standard deviation) with the View menu Ellipse radius command. We display the projection of the ellipsoid onto the cluster window.

## User shapes

We also maintain a two-dimensional user ellipse and a user-defined shape that are used to surround events and give them a user-defined class. The user ellipse is independent of the three dimensional class ellipsoids.

## Toolbar controls



There is a Toolbar across the top of the display. Each button has an associated tool tip that is displayed if the mouse pointer lingers for more than half a second over a button. The tool tip contains a reminder of the function of the button. The buttons are arranged in three groups: The group on the right controls the display of events in the cluster window.

### Code buttons



There is one button for each class code that is present in data. Each button displays an event colour and class code. The colour scheme is the same as for WaveMark data in a time view. Events are displayed for buttons that are down, events are hidden when buttons are up. Only visible events are modified by the commands in this dialog. The All button is a short-cut to display all the event codes. Right-click All to hide all codes. The ! button inverts the displayed class codes; all visible event classes are hidden, all hidden event classes are made visible.

### State buttons



The group on the left provide short-cuts for various menu commands. You will find the details of these commands in the View menu description, below. From left to right:

|                   |                                                                                                                           |
|-------------------|---------------------------------------------------------------------------------------------------------------------------|
| Autoscale         | Scale the image to keep everything visible.                                                                               |
| Link close events | Link events closer that a minimum time period with an arrow. Right click this button to open the Minimum Interval dialog. |
| Show axes         | Display $x$ , $y$ and $z$ axes in the display area.                                                                       |
| Use Time as Z     | Use event times as the $z$ axis for display purposes                                                                      |
| Density plot      | Display a density plot. Right-click to open the settings dialog.                                                          |

### Ellipse buttons



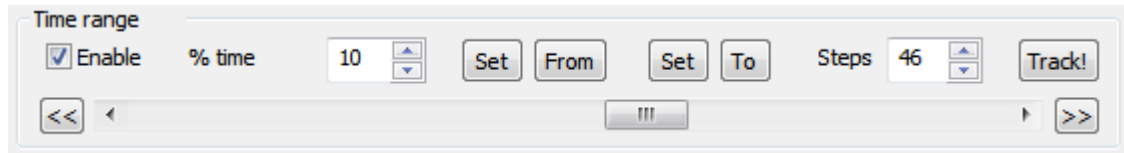
The central group controls the display of shapes that either describe the extent of a class of events or that are used to mark events as belonging to a class From left to right:

|                |                                                                                                                                     |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------|
| Class ellipses | Display ellipses for each visible class of events. Right-click the button to open a dialog that controls the size of the ellipse.   |
| User ellipse   | Display a user-defined ellipse. This is used to select events for coding and to restrict the events used in the interval histogram. |
| User shape     | Display a user-defined shape formed from straight lines. This is used in the same way as                                            |

the User ellipse. To create a User shape, Alt-click in the display area, then click for each new node and double-click to close the shape.

## Time range control

You can hide and show the Time range control area with the View menu Show Time Range command or with the `Ctrl+T` keyboard short-cut. This area restricts the number of displayed events by setting a time range that they must lie within. The % time range field sets the time range as a percentage of the time range spanned by all the events that were analysed. The thumb of the scroll bar is set to a size to match the time range.



When you check the **Enable** check box, the events displayed in the cluster window and used to build cluster statistics (and hence the class ellipses) are restricted in time. You can drag the thumb of the scroll bar to change the time range. If you click the `>>` or `<<` buttons the scroll bar steps through the data in the indicated direction, a second click stops the stepping. You can use this to see if the clusters change position with time.

When you change the time range, everything that depends on the displayed events changes to track the displayed time range. If you have an active interval histogram and you have selected automatic updates in the INTH settings dialog, the histogram will update to track the current time range.

If the number of displayed items of any class falls below the minimum required to calculate the class ellipses (4 in three dimensions, 3 in 2 dimensions), the class ellipse will not move until sufficient items are in the selected time range.

## Tracking clusters

If you have clusters that move with time you can use the **Track!** button to assign class codes automatically over a time range. Proceed as follows:

1. Move the thumb to the start of the time range to track and make your initial cluster assignments, then click the **Set** next to the **From** button to mark the start position.
2. Move the thumb to the end of the time range (this can be before or after the **From** position time) and click the **Set** next to the **To** button to mark the end position.
3. Set the **Steps** field to the number of intermediate steps to use in progressing from the **From** position to the **To** position.
4. Click **Track!** to class events based on the clusters at the **From** position. The **K Means** dialog will open and you can choose the method to use for normalising the data. Once this is done, click **OK** to start tracking.

The **From** and **To** buttons jump the scroll bar to the start and end of the time range. The tracking algorithm works as follows:

1. The clusters at the **From** position are assumed to be correct. The statistics of these clusters are calculated. All the events past the start position to the end of the time range are marked as code FF – which stands for unknown.
2. The time range is moved on by:  $(\text{To time} - \text{From time}) / \text{Steps}$  and we run the **K Means** algorithm on the new data and recalculate the statistics. The classes of events more than 3 times the Mahalanobis distance from the cluster centres are not changed.
3. Repeat step 2 until we reach the **To** time. Any events that are still marked as unknown are set to code 00.

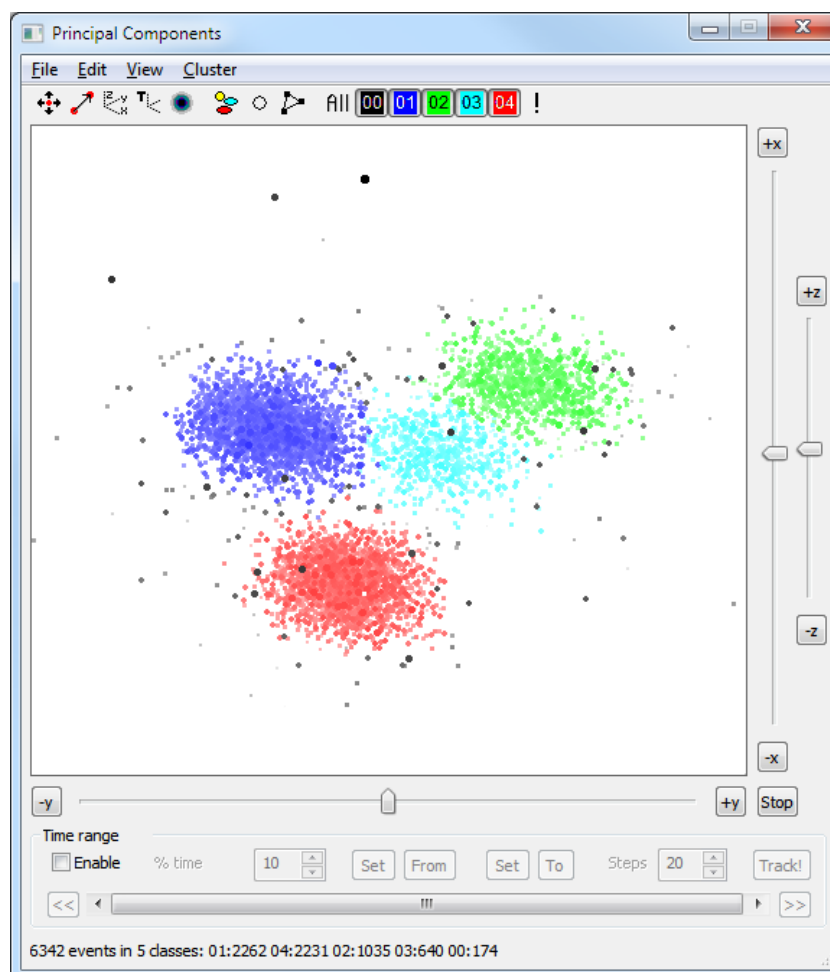
The larger the number of steps, the better the algorithm will cope with rapid cluster movement, but the longer the process will take. Because the algorithm is based on **K Means**, all the problems that **K Means** suffers from also apply. If you have periods where a class of events vanishes, or if new events appear in the middle of a time range, you may need to break down the tracking into multiple time ranges.

If there are clusters that do not move, you can greatly simplify tracking by coding these first (you may need to do this with the **Set Codes** command) and then hiding them.

## Identifying events

The mouse pointer changes to a cross hair when it is over the cluster window. To find out which event produces a particular dot in the window, centre the cross hair on the dot and click the mouse button and release. Spike2 will locate the nearest event to the centre of the cross hair and display it in the Edit WaveMark dialog. Cursor 0 will also move to the spike position in the associated time view. This does not work if you are on-line and the *always take spikes from the end of the file* button is down in the Edit WaveMark dialog.

## Scaling and shifting the cluster window



*Clustering dialog (click features for more information)*



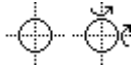
You can zoom in to an area in the cluster window by clicking and dragging with the mouse. While the mouse is down, the mouse pointer changes to a magnifying glass with a plus sign in the centre and a dotted rectangle shows the selected area. When you release the mouse, this area expands to fill the window. If you hold down the `Ctrl` key before you click and drag, the magnifying glass has a minus sign in the centre and when you release the mouse, the original area of the window is scaled to fit in the dotted rectangle.



You can drag the cluster window by holding down both the `Shift` and `Ctrl` keys before you click and drag the display. The mouse pointer is an open hand before you click and a closed hand when you drag.

## Rotating the cluster window

There are three sliders with associated buttons that rotate the cluster window around the x, y and z axes. Initially, x runs from left to right, y runs from bottom to top and z is out of the screen towards you. The small buttons at the ends of the sliders (labelled +x, -x, +y, -y, +z and -z) set a continuous rotation. Each time you click one it changes the rotation rate. The **Stop** button cancels continuous rotations. If the sliders are not visible you can use the `Ctrl+S` short-cut key to show them.



You can also rotate the cluster window using the mouse. If you hold down the `Shift` key and move the mouse over the cluster window, the mouse pointer changes to a cross-hair with a circle.

Now imagine that the events are in a large glass sphere and that you are looking at it through a square window. If you click and drag, with the `Shift` key down, this "grabs" the surface of the sphere and moves it in the direction that you are dragging.

When you rotate the cluster window, any displayed class ellipses also rotate. If the user ellipse is present it does not change.

### No z information

In some cases, the data resulting from the analysis will not have any z information. In this case, rotation with the x and y sliders and by clicking and dragging is disabled. You are still allowed to use the z slider to rotate the data around the z axis.

## Jitter the cluster window

Instead of rotating the view, you can apply a small circular jitter to the display with the Jitter command in the View menu. This allows you to rotate the display to best view the clusters, then apply a small jitter to help you visualise the clusters in three dimensions.

## Working with ellipses



You can select, drag and modify the shape of displayed ellipses. Click on or within an ellipse to select it; four resizing handles appear to show that it is selected. To select multiple ellipses, hold down the `Ctrl` key before you click. If you click in a region that is enclosed by more than one ellipse, the ellipse with the smallest area is selected.

To move a selected ellipse, click inside the ellipse and drag. To resize an ellipse, click on one of the handles and drag. The opposite handle is fixed while you drag. If you hold down `Ctrl` and drag, the ellipse resizes around its centre. If the display window scaling is different in the x and y directions you will find that the ellipses distort if you rotate them by dragging a handle. You can force the scales to be the same in both directions with the View menu **Equal scales** command.

When you select one or more ellipses, the status bar displays a list of the classes and the number of events in each class that lie within the selected areas. The class list is sorted with the largest class first.

You can change the class codes of the events inside or outside selected ellipses with the Cluster menu **Set Codes** command. As a short-cut, if you press the keys 0 through 9 with one or more ellipses selected, all events within the selected ellipses change class to match the key: 0 for class 00 up to 9 for class 09. You can also set the codes of all visible events with the `Ctrl+Shift+0` through `Ctrl+Shift+9` key combinations. Beware that Microsoft have grabbed `Ctrl+Shift+0` for IME language control in Vista and Windows 7. You can get instructions to defeat this [here](#).

Although you can change the positions and sizes of the class ellipses, such changes have no effect on the classes unless you use one of the methods discussed above to change the class codes. To restore the positions of the class ellipses, click the toolbar **SD** button twice: once to hide ellipses, and a second time to restore them to their original positions.

With x, y and z axes, class ellipses can only be calculated if the class contains at least 4 events and these events are not co-planar. If you are working with only x and y axes, you need at least 3 events and they must not lie on a line.

## Working with the user defined shape



If your clusters are not ellipsoidal, you may find it easier to select items with the user-defined shape. To start a shape, hold down **ALT** and click in the cluster window. Then move the mouse and click to add additional corners. To close the shape, click at or near the start point, or double click to add a point and then close the shape.

The user-defined shape behaves in similarly to the user ellipse. You can select and move it around the cluster window. You can select individual handles and drag them to adjust the positions of the corners. With the shape selected you can set classes with the **Cluster** menu **Set Codes** command, exactly as for the user ellipse.

## Online Clustering

When you open the clustering dialog while sampling data, you can choose to have new events added into the display as they occur (online update) or to run exactly as if you were analysing a file offline. You can enable and disable online update mode, set how far back in time to search for events, and limit the number of events to display with the **File** menu **Online Update** dialog.

If the **Edit WaveMark** dialog is in **At End** mode when the cluster dialog opens, the clustering starts in online update mode, and processes the most recent range of data as specified in the **Online Update** dialog. Otherwise, the range of data set for the **Edit WaveMark** dialog is processed. This initial analysis sets the parameters for online analysis. Changes in the **Edit WaveMark** dialog have no effect on online updates; you must use the **Reanalyse** command to pick up such changes.

### Principal component analysis

Principal component analysis is complex and time-consuming. It is not practical to re-analyse all the data for each added event. Online updates use the matrices created by the initial analysis or by the last **Reanalyse** command to map new spikes onto the screen. The added spikes do not change the principal components.

### Measurements

This method works identically online and offline.

### Cluster on correlation and Cluster on errors

These methods take a snapshot of the templates in the **Edit WaveMark** dialog when the cluster windows are created and each time you **Reanalyse** the spikes. The online update mode uses these snapshots, not the templates in the **Edit WaveMark** dialog.

## Menu commands

The clustering dialog has its own menu. All references to menu items in this section of the documentation are to the dialog menu, not the main **Spike2** menu. Most menu items have short-cut keys that allow immediate use; you can also right-click in the dialog to open a context menu that duplicates many of the menu commands.

- |                     |                                                                                                                                                                               |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>File menu</b>    | Apply changes, restore the original class codes from the data file, reanalyse the data, create interval histograms of the data and display the principal component waveforms. |
| <b>Edit menu</b>    | Undo class changes, copy an image, copy data as text and edit interval histogram settings.                                                                                    |
| <b>View menu</b>    | Everything to do with the appearance of the display and control over rotation, automatic scaling and ellipse and dot sizes.                                                   |
| <b>Cluster menu</b> | Commands for automated data clustering and setting class codes.                                                                                                               |

## File menu

The file menu contains commands to apply changes, restore code, reanalyse the data, generate an interval histogram, display the principal components and select the principal components to use when clustering.

## Apply changes

This copies the current state of the classes in the clustering window into the data file. Any events that were skipped during measurement analysis are treated as having a class code of 00. For this to work as expected you must not have changed the marker filter for the data channel since extracting the clustering values. If you do not use this command, all class changes you make in this dialog are lost when the dialog closes.

In addition to changing the events in the data file, the associated spike shape dialog will recalculate its templates based on the class assignments of the events in the marker filter and current time range set in the dialog. If you are working on-line, the newly created templates are copied to the 1401 for immediate use.

## Restore codes

This command sets all the class codes to match the data file. You can use this to restore values after an unsuccessful clustering attempt. Another use is to pass class information between the measurement and PCA clustering dialogs if you have both open at the same time; use Apply changes in one and Restore codes in the other.

## Reanalyse

This command opens the Principal Components, Cluster on measurements, Cluster on Correlations or Cluster on Errors setup dialog so that you can repeat the analysis.

## Online Update

This File menu command is enabled if you are clustering a channel that is being sampled. There are four fields you can set:

### Update interval

This sets the minimum gap between updates of the cluster window in seconds. A value of 0 updates the window as often as possible.

### Maximum events to display

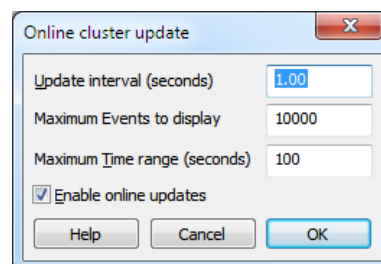
This limits the number of events to display in the window. At high event rates, if you set a large number and a short update interval, Spike2 may become slow to respond as it is spending a lot of time recalculating the display. You can delete all the spikes in the display with the Edit menu Delete online spikes command.

### Maximum Time range

This field limits how far back from the current time to search for events. The combination of this and the Maximum events to display field limits the number of displayed events.

### Enable online updates

You can enable and disable online updates here. If you disable them, the Reanalyse command will process the time range set for the Edit WaveMark dialog, or the entire file if the Edit WaveMark dialog is in At End mode.



## Interval histogram

This command creates an interval histogram of the events in the clustering window. If an interval histogram of the events already exists, the previous values in the result view are replaced by a new analysis. Each class code present in the clustering window generates a separate channel in the histogram. The histogram bins are coloured to match the class colours except for code 00, which is filled with white rather than black. You can set the histogram width and bin width with the Edit menu INTH Settings command.

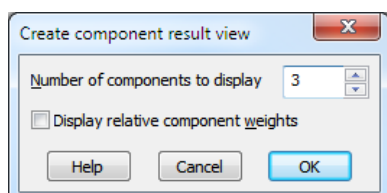
### Minimum interval cursor

A vertical cursor is added to the interval histogram at the position of the minimum interval set for the clustering. If you drag the cursor in the histogram, this will change the minimum interval, and if the cluster display is set to link events closer than the minimum interval, the display will update on each change.

### Restrict events with a user shape

You can restrict the events that are displayed in the INTH by enabling the user ellipse of the user shape and selecting it. This can be used to check that the events within a shape are not too close to each other.

## Display Principal Components

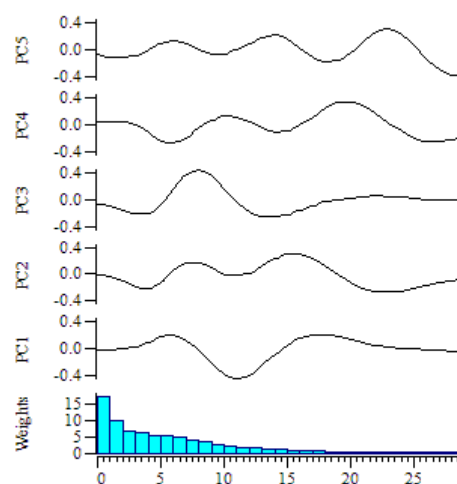


This command is enabled for Principal Component Analysis. It creates (or updates if you have already created it) a result view holding the principal component waveforms and optionally the relative weights of each component. You can choose the number of components to display. The display order is in terms of their significance, based on their contribution to the variance of the original data. You will usually find that only the first 3 or four components make a useful contribution to clustering.

If you choose to display the weights, the x axis of the result view is the component number for the weights which corresponds with data points for the principal component waveforms. If you do not display the weights, the x axis is set to milliseconds with zero time representing the trigger point when the data was captured. If the spike data has multiple traces, each component shows all the traces, in order.

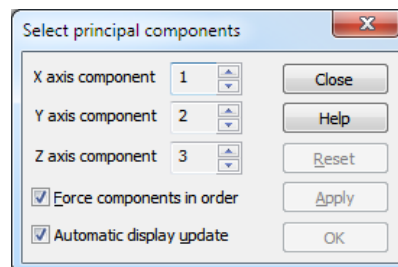
The tail end of the weights usually appears as a smooth curve trending to zero. The principal components that correspond to these weights are generally just noise.

The above assumes a waveform-based analysis. Analyses of amplitudes or amplitude ratios will not make any sense when drawn as waveforms.



## Select Principal Components

This dialog sets the principal components to display. **Reset** selects components 1 to 3; these are usually the best ones to choose.



### Force components in order

Check this box to make sure that the X component has the lowest number and the Z component has the highest.

### Automatic display update

Check this to update the display on every change; this can be slow if you have a very large number of events. The Apply button can be used if you disable automatic updates.

## Close

This closes the dialog. It does not save any changes you have made to the spike classes.

## Edit menu

This contains the following commands:

### Undo

This undoes the last change made to the class codes. At the time of writing only one change is saved. In the future we may allow more than one undo.

### Copy

This command copies the cluster window to the clipboard as a bitmap. The bitmap image is always created with z buffering enabled, so events that are further away from the viewer are hidden by events that are closer. This may differ from the screen image where the z buffer is only used if selected because it slows down drawing.

### Delete online spikes

This command and the shortcut key **Del** are enabled when you are working online. It deletes all the spikes from the cluster.

### Copy As Text

Copy the current cluster information to the clipboard as text.

### Copy Cluster Values

This copies the clustering information for all visible cluster classes to the clipboard as text. The columns of text are separated by tab characters and the titles are surrounded by double quotes, which should make the data easy to import into other programs. The output columns contain: the cluster code, the number of items in the cluster, then for each axis the cluster centre and the sigma value (square root of the variance about the centre) and the mean cross products about the centre (these would be 0 if the clusters were aligned with the axes, equivalent to being circular). The values for the Z axis are omitted if there is no Z axis.

| "Cluster" | "N"  | "X"      | "SigmaX" | "Y"      | "SigmaY" | "Z"        | "SigmaZ" | "XY"      | "XZ"     | "YZ"    |
|-----------|------|----------|----------|----------|----------|------------|----------|-----------|----------|---------|
| 00        | 17   | -1.47037 | 0.986724 | 0.234567 | 0.950138 | 0.046902   | 1.02006  | -0.434    | 0.1411   | 0.49357 |
| 01        | 2348 | -1.01898 | 0.462661 | 0.400486 | 0.266091 | 0.221326   | 0.80008  | 0.000917  | 0.00818  | 0.01794 |
| 02        | 1036 | 1.48261  | 0.414779 | 1.39831  | 0.288466 | -0.329622  | 1.18115  | -0.001983 | -0.0379  | 0.10251 |
| 03        | 645  | 0.742498 | 0.38566  | 0.446943 | 0.278826 | -0.29484   | 1.00934  | -0.005489 | -0.02464 | 0.04993 |
| 04        | 2218 | 0.181543 | 0.361582 | -1.20886 | 0.279064 | 0.00504552 | 1.03087  | -0.001484 | -0.00329 | 0.08619 |

In this example, the mean cross products are small compared to the variance (apart from cluster 00), meaning that the data va



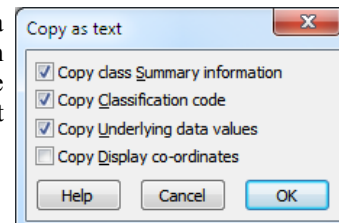
## INTH Settings

Change the settings for the interval histograms of each of the cluster classes.

## Copy As Text

This menu command opens a dialog from which you can copy clustering data to the clipboard as text in a spreadsheet-compatible format. The output is in columns separated by tabs, with text enclosed in double quotes. There is one line of output for every visible event in the cluster window. The text output has the form:

```
6342 events in 3 classes: 01:3300 02:3002 00:40
"Time" "Code" "X" "Y" "Z"
0.00016 "02" -0.0107435 0.00288545 -0.0126408
0.03768 "03" -0.0151009 0.00494189 0.00245431
0.17096 "02" -0.00890594 0.00563056 -0.0174058
```



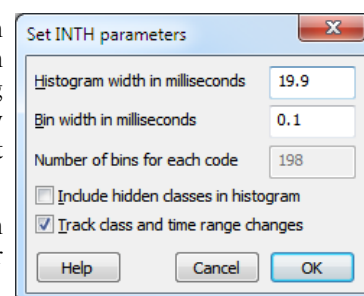
The time of each event is always output in the first column. You can also choose to copy the following, which are described in the order that they are added to the output line:

- Summary information** This is the same as the line of text displayed at the bottom of the cluster window.
- Classification code** Check this box to output the class code for each event, with the column title "Code".
- Underlying data** These are the x, y and z values that resulted from the principal component analysis or the measurements. The columns titles are "X", "Y" and "Z". If there are no z values, the z column is omitted.
- Display co-ordinates** These are the underlying data values rotated into display co-ordinates. If there is no rotation these are the same as the data values. The x and y values set the event positions in the cluster window. The z value is used for the z buffer. The titles are "View X", "View Y" and "View Z". If there are no z values in the original data, the "View Z" column is omitted.

## INTH Settings

This menu command opens a dialog that sets the parameters for creating an interval histogram based on the events in the cluster window. The histogram has multiple channels, one for each event class. When you close the dialog with the OK button, the interval histogram is created or updated if it already exists. The File menu Interval histogram command also uses the values set in this dialog.

You can set the desired histogram width and the width of each bin, both in milliseconds. The number of bins in the histogram is also displayed for your information, but this field cannot be edited.



If you set the Include hidden classes in histogram check box, all event classes are processed and displayed. If the check box is clear, only visible classes are included and the displayed classes will change when you show or hide classes in the cluster window.

If you check the Track class and time range changes check box, any operation that changes the visible events or the event classes causes the interval histogram to recalculate.

## View menu

This menu (plus suitable entries depending on your current selection) can also be activated as a context menu by right-clicking the mouse in the cluster window.

## Autoscale, Equal Scales, Rescale

The events to be drawn each have an x, y (and possibly z) position. When you rotate the display, these positions are multiplied by a rotation matrix to generate an x', y' and z' display co-ordinate. If your data had very different ranges of x, y and z (for example when using measurements and not normalising the result), as you rotate the display, the width and height of the result will change dramatically. To counter this, we can also apply a display scaling that looks at the range of values in x' and y' and scales the result so that they fill the available space.

### Autoscale

When you rotate the cluster window you may find that the data tends move out of the viewing area. Turn this option on to rescale the view to fit all visible events, axes and class ellipses in the window. The view rescales automatically if you change the displayed classes, the displayed time range or the window rotation. If the Equal scales option is set, the larger of the two display axes sets the scaling for both axes.

The state of Autoscale is indicated by a button in the toolbar.

If your interest is purely in the clusters and not at all in the actual value used for clustering, you can often remove the need for scaling by checking the normalise box in the setup dialog for the analysis you used to create your cluster data. This forces the x, y and z values to have a zero mean and a unity variance.

### Equal Scales

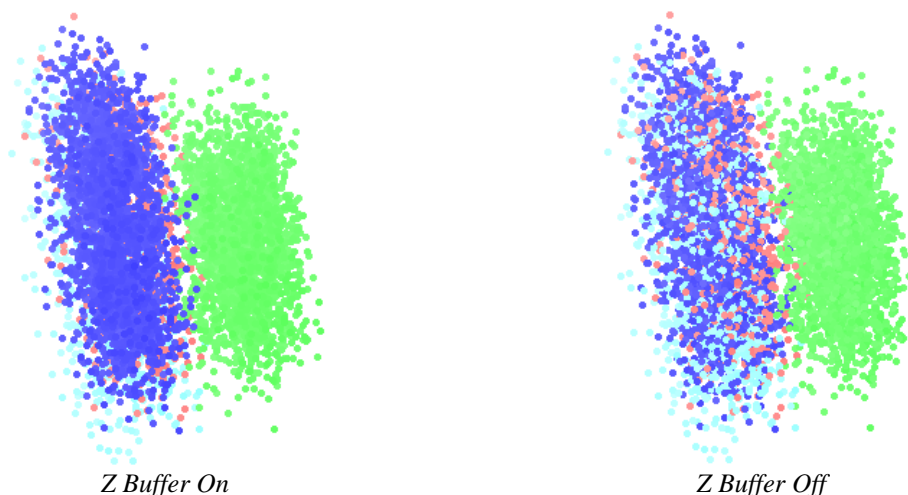
If you enable this option, the cluster window is scaled so that the width and height of the window have the same numeric size. This makes ellipses easier to manage, as they will preserve their shape when rotated. The more different the scaling of the x and y display axes, the more the ellipses will distort when rotated. Put another way, whichever of x' or y' has the greater numeric range sets the scaling for both the x and y directions.

### Rescale

This command scales the display window so that all the displayed event data, axes and any class ellipses are visible. If the Equal scales option is set, the larger of the two display axes sets the scaling for both axes. If axes are enabled, the axis origin is also displayed. You would only need to use this command if Autoscale were disabled.

## Z Buffer

With this option on, events nearer to the viewer draw on top of events that are further away. With the Z buffer off, events are drawn in time order. Turning this option on increases the illusion of three dimensions, but at the cost of drawing more slowly. If you have a very large number of events you may be able to improve the drawing speed if this option is off. The keyboard short-cut for this option is `Ctrl+B`.



The use of the Z Buffer is recommended if you use the Colour Fade with Z or variable dot size options. You should only need to turn it off if you have a very large number of points to display and the display response has become too sluggish to be useful.

When you copy the image to the clipboard, this option is always turned on. The previous state is restored once the bitmap has been copied.

## Dot size

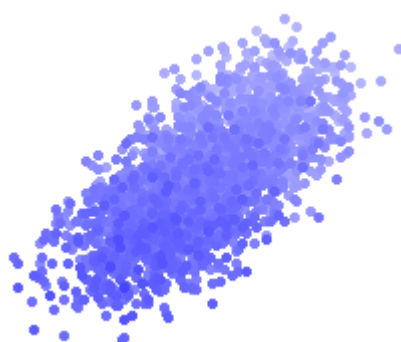
The events in the cluster window can be drawn at five sizes, numbered 0 to 4; the larger the size, the larger the dot and the longer it takes to draw the events, especially if the Z Buffer is enabled. You can also choose *Varies*, which draws each event at a size that depends on the notional distance of the event from the viewer.

If you are using circular dots, you will get the best effect with size 4. Smaller sizes do not produce convincing circles! Sadly, using colour shading to make small shapes look more circular works well for isolated dots, but tends to produce a nasty "speckled" effect when dots overlap, so we have not implemented it.

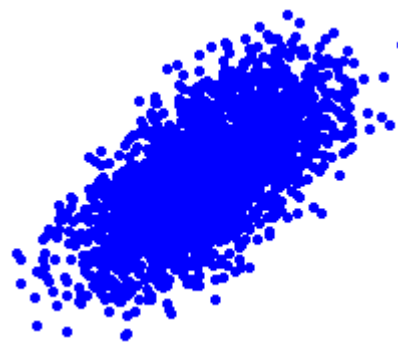
The *Varies* method can be effectively combined with *Colour Fade* and the Z Buffer to enhance the impression of three dimensions.

## Colour Fade with Z

Each displayed event has a notional x, y and z position with respect to the viewer. The x and y positions are used to set the screen position. If you enable this option, the z position is used to fade the event colour to the background colour as the event becomes further away from the viewer. The scaling is based on the maximum distance between the nearest event and the furthest (in the z co-ordinate) at the last time that the display was scaled. If *Autoscale* is enabled, this will always be set for the current data. If *Autoscale* is not enabled, this is the scale set by the last *Rescale* or *Equal Scales* operation. The keyboard short-cut is `Ctrl+F`.



*Colour Fade On*



*Colour Fade Off*

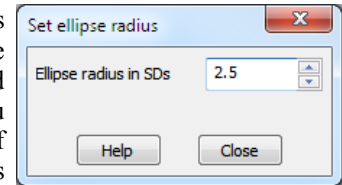
There is a small time penalty for using this option, which might become significant if you have a very large number of data points. This option works best with the Z Buffer enabled.

## Circular Dots

You can choose to display square dots or circular dots. Circular dots become non-square at size 3, and are really only successful at size 4 (well you try making a circle out of solid colour pixels at sizes of 3x3 pixels or less!) Circles are slower to draw than squares, so if you have a huge number of points (tens of thousands) you may find that drawing as squares or at size 0 is faster. The short-cut for this command is `Ctrl+L`.

## Ellipse radius

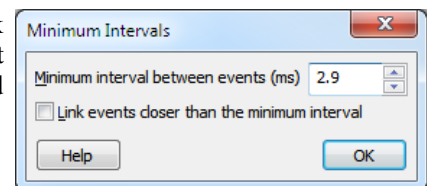
This command opens a dialog in which you can set the size of the class ellipses that are drawn around each class cluster. The size is set in terms of the Mahalanobis distance, which is a multi-dimensional analogue of the standard deviation. If the data were normally distributed around the mean position, you would expect 68% of the events to be within the Mahalanobis distance, 95% of the events to be within twice this distance and 99% to lie within three times this distance.



You can set the distance to a value in the range 1.0 to 4.0 times the distance. The default is 2.5 times, which seems to be about right. If the class ellipses are visible, any change made in the dialog has immediate effect.

## Minimum Interval

One way to check that codes set by clustering are reasonable is to look at the time intervals between events in the same class and verify that events are not too close together. The Minimum Interval command opens a dialog with the following fields:



### Minimum interval between events (ms)

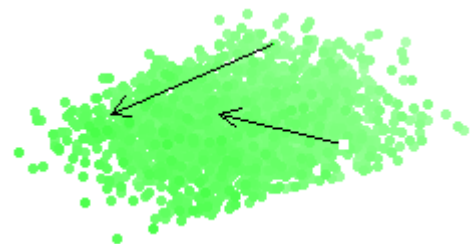
This field sets the minimum acceptable interval, in milliseconds, between events with the same class code. Setting a value here has no effect unless the next field is checked.

### Link events closer than the minimum interval

If you check this box, pairs of events that are closer than the minimum interval field are linked by a line with an arrow pointing from the earlier event to the later event. You can left-click on the events to move to them in the Edit WaveMark dialog. The arrows are drawn on top of the data, even if they belong to events that are buried inside a cluster.

### Find Short Interval

If there are a lot of events, even though the arrow that links them is drawn on top of everything else, it can be difficult to find the event without zooming in. There is an easy way. Click within a few pixels of an arrow or right-click within 100 pixels and select the Find Short Interval command in the pop-up context menu. This will then find the nearest line to the click point and then make the event that starts that interval the current event. This event is displayed in the associated Edit WaveMark window, and if that dialog is set to scroll the time window, the time view will also scroll to show the event.



*Short interval arrows*

### Link to INTH window

If you display the INTH window, a vertical cursor appears at the minimum interval you set in this dialog. Changes made in this dialog set the cursor position. You can also drag the cursor in the INTH window to change the minimum interval.

## Use Time as Z

You can choose to display the clustering data using Time as the Z axis. This is not useful when clustering data, but can help in visualising changes with time. To do this, the times of all the data points are mapped into the range -1 for the oldest event to +1 for the newest. Applying and removing this option can take a noticeable time if you have a very large number of events. This option is not allowed if you are in online update mode, or if you have had an online update.

If you select this mode and the axes are visible, the Z label on the axis changes to T.

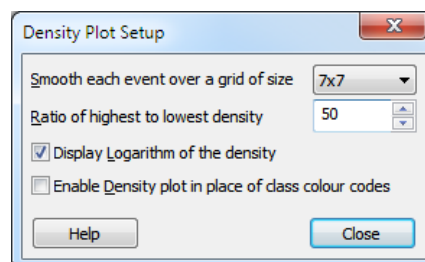
The short-cut for this is  $T$  (or  $\text{Shift}+\text{Ctrl}+T$  which may be removed in future).

## Density Plot

There are two basic display modes: Density Plot and Class Colours. This command chooses between the two modes. In the Class Colours mode, each event is drawn as a dot in the colour of the event class. In Density Plot mode, each pixel of the display is drawn in a colour that shows then number of events at that point in the display.

## Density Settings

This command leads to the Density Plot Setup dialog, which controls how the event density is converted into a density display.



### Smooth each event over a grid of size

To convert the events from individual points to a smoothly varying density plot, each event is spread over a square grid of pixels that is centred on the event and the result of this is summed over all the events. You can choose from a 1x1 grid (no smoothing) up to a 7x7 grid.

### Ratio of highest to lowest density

The highest density found is allocated to one end of the density colour scale. This figure sets lowest relative density to the highest density that is visible as a contrast with the background. Smaller numbers will make individual outlier events merge into the background.

### Display Logarithm of the density

If all your clusters have similar densities, then it makes sense to map densities to your colour map linearly. However, if you have clusters of widely differing densities, it can make more sense to display the logarithm of the density.

### Enable Density plot in place of class colour codes

This is exactly the same as the Density Plot command and allows you to swap between a display of class colours and density from within the dialog.

## Density Colour Map

This command allows you to choose the colour scale that is used to indicate density. This dialog is exactly the same as for the sonogram colour scale as described for the View menu.

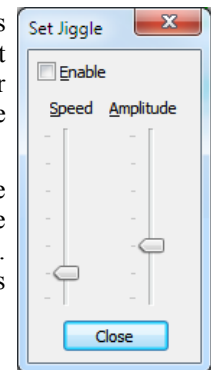
## View along axis

This command (enabled with three dimensional data only) display a pop-up menu with three options: X, Y and Z. Choose one of these options to rotate the cluster view to point one of the three data axes upwards and cancel any display rotation. It does not cancel any display jiggle. The keyboard shortcuts for these commands are x, y and z.

## Jiggle Display and Jiggle Settings

With three-dimensional data, it is sometimes easier to visualise the clusters if the display is "jiggled" around the current view angle. If you enable the display jiggle, this is equivalent to using the mouse to rotate the display in a circle around the centre of the displayed cluster area. You can enable and disable this rotation with the Display Jiggle command or with the `Ctrl+J` keyboard shortcut.

The Jiggle Settings dialog gives you additional control over the jiggle. The Amplitude slider sets the radius of the notional circle and the Speed slider sets how fast to rotate around the centre of the circle. The Enable check box starts and stops the display jiggle. The keyboard shortcut to open the dialog is `Ctrl+Shift+J`. The image to the right has been adjusted for height; the dialog is much taller than shown here.



## Clear rotation

This command restores the cluster window to its original, non-rotated state and removes any continuous rotation set by the buttons around the rotation slider controls. It also cancels any display jiggle.

## Show Sliders

You can choose to show or hide the slider controls and associated buttons that rotate the clustering window around the x, y and z axes. The dialog size will grow, if required, when you turn on the sliders.

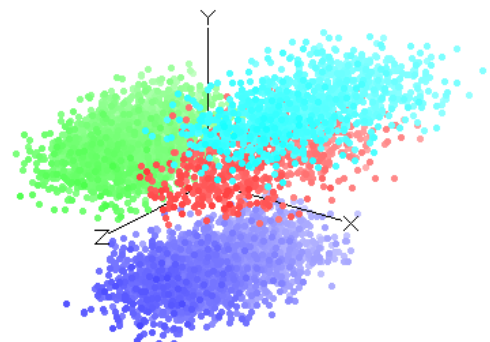
## Show Time Range

The Time range control area can be shown and hidden as required. The dialog will change size when you show and hide this area. This area allows you to display events that fall within a restricted time range and track how the clusters change with time.

## Show Axes

You can display axes in the cluster window. They are labelled X, Y and Z and are drawn along the principal axes of the measurements. The keyboard command to show or hide the axes is `Ctrl+X`.

If the Z buffer is enabled, the axes are drawn with true depth, that is events that are further away will be behind the axis lines and events that are nearer will be in front, as in the image. The X, Y and Z used to label the axes are always drawn on top of the events. If the Z buffer is not enabled, the axes are always drawn on top of the data. In Density Plots, the axes are always under the data.



*Axes with the Z Buffer active*

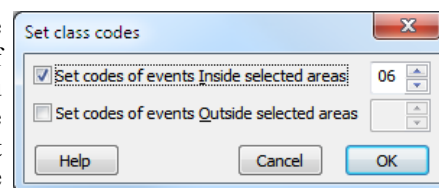
## Cluster menu

This menu contains commands related to manual and automatic assignment of class codes to event clusters.



## Set codes

This command is enabled when there are selected shapes in the cluster window. It opens a dialog in which you can set the codes of all events that lie inside and/or outside selected shapes. When you open the dialog the inside code is set to the most common code of the selected events unless this code is 00, in which case the lowest unused code is set. The outside events code remembers the last code you set.



The check boxes control which set of events has its codes changed. The edit boxes on the right set a code to apply as two hexadecimal digits (0-9, a-f or A-F). If you set a single character in the field, a 0 will be inserted in front of it. Leaving the field blank is the same as clearing the check box and no class codes will change.

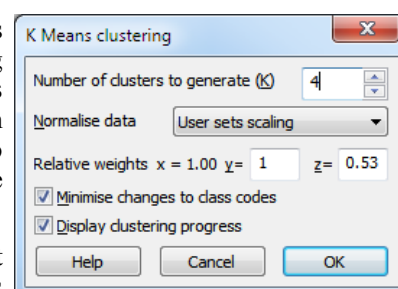
Click OK to change the displayed events. This has no effect on the original spikes in the data file. You must use the File menu Apply command to change their class codes. You can undo this command with `Ctrl+Z`.

## Clear all visible codes

This command is here as a quick way to set the class codes of all visible events to 00. It has a short-cut key combination `Ctrl+Shift+0`. In fact, you can set all visible events to any code from 00 to 09 using `Ctrl+Shift+0` through `Ctrl+Shift+9`. You can undo the effect of these commands with `Ctrl+Z`. Beware that Microsoft have grabbed `Ctrl+Shift+0` in Vista and later for IME use, see here for how to defeat this.

## K Means from existing and K Means

These commands open a dialog in which you can assign events to classes automatically. The **K Means from existing** command uses the existing cluster centres as the seeds for the K Means algorithm. The **K Means** command runs the algorithm 10 times (or until you stop it) with random seeds and chooses the result that has the best ratio of cluster separation to cluster size. You can find details of the algorithm at the end of the chapter.



The K Means algorithm is a well-known and fast clustering method that is effective when the number of clusters is already known, the clusters are spherical and are of a similar size. Put another way, if the clusters are not spherical (or cannot be made spherical by scaling) or are of very different sizes, KMeans will not give useful results.

### Number of clusters to generate

This field sets the number of clusters to generate. It is preset to the number of visible, non-zero class codes in the data set. We allow you to set from 1 to 20 clusters. This field is disabled if you have selected the **K Means from existing** command.

### Normalise data

This field sets the strategy to use to make the clusters spherical. You can choose from:

- |                       |                                                                                                                                                                                                                                                                                                                                  |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| None                  | No scaling: using the measured values directly. This is equivalent to the <b>User sets scaling</b> option with all weights set to 1.                                                                                                                                                                                             |
| Use existing clusters | The relative scaling of the x, y and z data is deduced from the shape of the current class clusters.                                                                                                                                                                                                                             |
| Use visual scaling    | We assume that you have arranged the display so that the clusters appear circular. The data is clustered based on the visual appearance of the data in the window.                                                                                                                                                               |
| User sets scaling     | Extra fields appear in which you can set the relative weights to give to the y and z values relative to the x values. For principal component analysis, these fields are initially set to values based on the relative weights of the components. For measurement-based clustering, the relative weights are initially set to 1. |

### Minimise changes to class codes

If you check this box, the event class codes are chosen to maximise the number of events that keep the original code. If you do not check this box, class codes are assigned so that the lowest free codes are used. This is always checked for K Means from existing.

### Display clustering progress

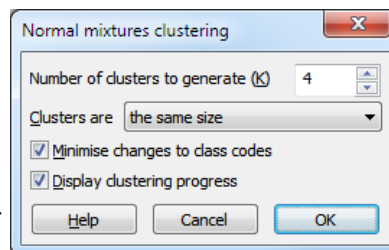
Check this option to follow the clustering in detail. The K Means command updates the display after each iteration regardless of this setting. This option slows analysis.

### Running the command

Click OK to run the command. This may take some time if there is a large number of events and K is large. When the analysis is complete the command calculates the ratio of the average separation of the clusters to the average cluster size. This is displayed in the status line as the J3 value.

## Normal Mixtures from Existing and Normal Mixtures

These commands open the Normal Mixtures dialog in which you can assign events to classes automatically. The Normal Mixtures from existing command uses the existing clusters as the seeds for the Normal Mixtures algorithm. The Normal Mixtures command runs the algorithm 10 times (or until you stop it) with random seeds and chooses the result that has the maximum likelihood of fitting the data.



The Normal Mixtures algorithm assumes that the probability density of data points around each cluster centre follows a multivariate normal distribution. This is a more general approach than the K Means algorithm, as it does not require spherical clusters and can often give results that seem more intuitively correct. However, it is a more complex algorithm than K Means and takes noticeably longer to run.

### Number of clusters to generate

This field sets the number of clusters to generate. It is preset to the number of visible, non-zero class codes in the data set. We allow you to set from 1 to 20 clusters. This field is disabled if you have selected the Normal Mixtures from existing command.

### Clusters are

You can use this field to place restrictions on the clusters. The choices range from least to most restrictive. You can choose from:

- |                                   |                                                                                                                                                                                                                                                                                                                 |
|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Independent                       | The clusters are independent of each other and have different sizes, shapes and orientations. This is the most general setting and takes the longest to run. It is usually most successful when run with existing clusters as the starting point. Using random starting points can generate unexpected results. |
| the same size                     | The clusters are assumed to be the same size and shape. This is often true, especially with principal component data, so this is the default value.                                                                                                                                                             |
| the same size and axially aligned | The clusters are assumed to be the same size and shape, and the principal axes of the cluster shapes are aligned to the x, y and z axes of the data.                                                                                                                                                            |
| the same size and spherical       | The clusters are assumed to be the same size and shape and are spherical. This is the most constrained setting and is the most similar to KMeans clustering.                                                                                                                                                    |

### Minimise changes to class codes

If you check this box, the event class codes are chosen to maximise the number of events that keep the original code. If you do not check this box, class codes are assigned so that the lowest free codes are used and ordered so that classes with more events have lower codes. This is always checked for the Normal Mixtures from existing command.



## Display clustering progress

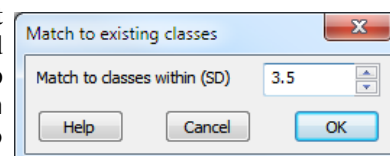
Check the box to follow the clustering in detail. The Normal Mixtures command updates the display after each iteration regardless of this setting. This option slows analysis.

## Running the command

Click OK to run the command. This may take some time if there are many events and the number of clusters is large. During analysis, a progress bar gives you the option to stop analysis early. When the analysis is complete the command displays the log likelihood value per point for the solution. See the discussion of the Normal Mixtures algorithm for more information.

## Match to classes

This command opens a dialog from which you can use the current cluster statistics to assign visible events to the nearest visible cluster and mark outliers as code 00. The typical use of this command is to clean up manual cluster assignments in situations where the K Means from Existing or Normal Mixtures from Existing commands would make too drastic a change.



The basis idea is that each visible event is assigned to the cluster that it is most likely to belong to. The shape of each cluster is taken into account, so a cluster need not be spherical or aligned with the x, y or z axes. Probability is measured in terms of the Mahalanobis distance, which is the multi-dimensional equivalent of standard deviation.

The Match to classes within field sets the maximum separation in terms of the Mahalanobis distance between a cluster and an event, for the event to be considered as a possible cluster member. Events that do not belong to any cluster are given code 00. The matching does not pay any attention to the number of items in each class (unlike the Normal Mixtures from Existing command). The matching does take into account the fact that the class ellipsoids may not be aligned with the x, y and z axes (unlike the K Means from Existing command).

## Getting started with clustering

We would suggest that you start by trying out principal component analysis, as this avoids you making subjective decisions about which features of your data are more important than others. It will also tend to produce clusters that are suitable for use with automatic clustering methods.

In the Edit WaveMark dialog choose the Analysis menu Principal Components command. This will open the cluster dialog and display the data, ready to cluster. It is a good idea to check that visible clusters are stable over time by using the Time range control area. There are three basic ways to cluster the data:

- |                |                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Automatic      | If you have reasonably well-defined clusters that are, or can be made more or less spherical by scaling, and that contain similar numbers of events, then the K Means command will be able to separate them for you. This will often be the case with principal component values.<br><br>If K Means does not make sensible choices, you should try the Normal Mixtures algorithm. This is slower, but can give better results. |
| Manual         | If the data is chaotic, the only sensible course may be to manually position ellipses or user-defined shapes and set codes. You can position an ellipse on a cluster centre by right clicking at the centre and selecting a small, medium or large ellipse from the context menu.                                                                                                                                              |
| Semi-automatic | If the automatic methods almost do what you want, you may be able to help them by marking the centres of clusters manually, then using one of the ...from existing commands to iterate from your centres.                                                                                                                                                                                                                      |

Whichever method you use, you may also wish only to mark events that you feel "certain" about. You can set

events that are too far from the cluster centres to code 00 with the Match to classes command.

Remember that changes made in the clustering dialog have no effect on the original data until you use the File menu Apply command.

---

## K Means algorithm

The K Means algorithm has many variants, but the basic idea is based on the following iteration sequence.

1. Given the current set of K cluster centres; assign each event to the nearest centre.
2. Recalculate the K cluster centres based on the new assignments.
3. If the centres changed in the iteration, go back to 1, else done.

The algorithm always converges, but there is no guarantee that it converges to the optimum solution. The result depends on the original cluster centres. The `K Means for existing` command takes the current set of cluster centres as the starting point. The `K Means` command assigns events to clusters randomly before starting the algorithm and repeats the entire procedure 10 times. It chooses the solution that maximises the variance of the distance between all clustered points and the centre of gravity of all the points divided by the variance of the distance between each point and the centre of its cluster. The better the clusters are separated, the higher the value. This measure is known as J3 in some literature.

Once we have iterated to a solution the next task is to remove outlier points. We generate the statistics for each cluster and based on the assumption that the data follows a multivariate normal distribution, we set all points that lie more than 3 times the Mahalanobis distance away from the centre of the their assigned cluster to have class code 00. We then run the K Means algorithm again, starting with the current centres. This usually converges in one or two iterations because removing outliers should make little difference to the distributions.

### Limitations of KMeans

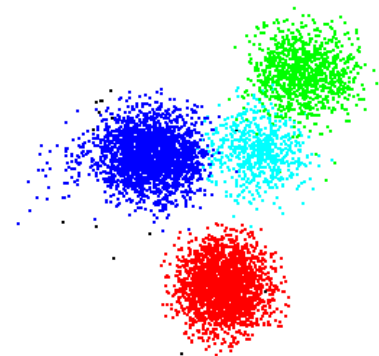
The KMeans algorithm assumes that the clusters are spherical and similar in size. The clustering is based on the distance between a each point and the centre of each cluster. It is often the case, for example with clustering based on measurements, that the scaling of the axes are very different. You can compensate for this to some extent by normalising the data before clustering (shifting and scaling so that the data on each axis spans a similar range about zero). However, this will probably not make the clusters spherical.

Assuming that the clusters are more or less the same size and shape, they can be made spherical by stretching or shrinking the y and z axes relative to the x axis. This is done by applying weighting factors to the y and z axis data (the x axis data is assumed to have a weighting factor of 1.0). When data comes from the Principal component analysis, we guess the weights based on the results of the analysis. When data comes from measurements, the weights all start as 1.0 and you can adjust them by hand. You can also choose to weight the data based on how it looks on the display, or apply weights calculated so as to make the existing clusters as spherical as possible, or to apply no weighting.

The KMeans algorithm also assumes that the x, y (and z) values are independent. If they are not, your data may look like ellipsoids that are not aligned with the x, y or z axis. If your data looks like this, you should probably be using the Normal Mixtures algorithm for clustering.

### J3 clustering measure

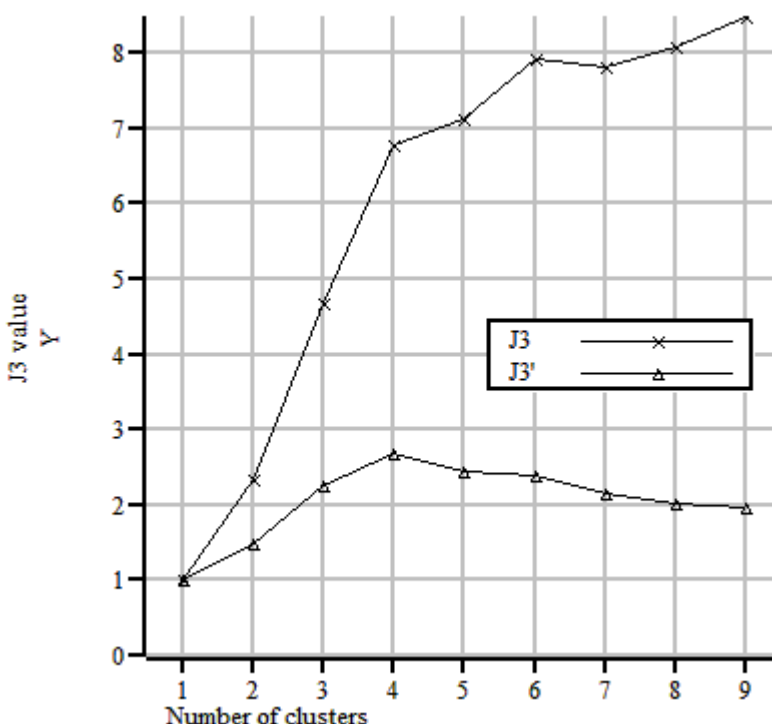
The J3 measure is displayed at the bottom of the clustering window after running the KMeans algorithm. It measures how compact the clusters are compared to the overall spread of the data. In this example there are 4 fairly clear clusters (although, at a stretch you could merge the two in the upper right corner together to make three clusters). The J3 values when analysing the data with different numbers of clusters are given in the graph. The data was normalised using the existing cluster sizes, so we repeated the KMeans process until the J3 value stopped changing.



The J3 value is given by  $J2/J1$  where J2 is the sum of the squares of the distances of all clustered points from the centroid of all the points and J1 is the sum of the squares of the distances of each point from the centroid of the cluster it belongs to. Dividing the data into more and more clusters reduces J1, so J3 increases with cluster count

In this case, the J3 value increases sharply from 1 to 4 clusters, then settles down at a fairly constant slope. Ideally, we would like a measure of clustering efficiency that would be a maximum at the most likely number of clusters.

We can estimate the value of J3 when data is divided up into more clusters than actually exist with the following simplistic argument. Consider the case where there are N points spread at a constant density through a  $m$  dimensional space. If this is divided up into  $k$  clusters of  $N/k$  points of similar (but arbitrary) shape, the radius (linear size) of each cluster will be proportional to  $(N/k)^{1/m}$  and the variance of the distances from the centre of each cluster (J1) will be proportional to  $(N/k)^{2/m}$ . By the same argument, the J2 value will be proportional to  $N^{2/m}$  with the same constant of proportionality, so J3 is given by  $k^{2/m}$ . What we are saying is that in a case where there is no way to divide data up into clusters (because it is spread through the space at a constant density), we know what the graph of J3 looks like.



In addition to J3, we also give the value J3', being J3 divided by  $k^{2/m}$ . This scales J3 by the value you would expect if all the points were uniformly distributed. In our case, we have  $m=2$  (x and y axes) or  $m=3$  (x, y and z axes). However, if one dimension makes no, or only a small contribution to the sorting, the effective number of dimensions is reduced so the J3' value can only be taken as a guide.

You can see that with this data, the J3' value peaks at 4 clusters, indicating that this is likely to be a reasonable interpretation of the data.

### Normal Mixtures algorithm

The Normal Mixtures algorithm assumes that the density of events around a cluster centre follows a multivariate normal distribution and that the number of clusters (K) is known. Clusters may be independent, or have the same shape, or have the same shape and be axially aligned or be spherical. We calculate the probability that an event belongs to a particular cluster based on the distances from each cluster, the shape of each cluster and the number of events in the cluster.

The algorithm is adapted from the Normal Mixtures algorithm described in *Clustering algorithms* by P.A. Hartigan, published in 1975 by Wiley, ISBN 0-471-35645-X. This process iterates towards a local maximum in the probability density of the set of events given the clusters. The value we return is the log of the probability density divided by the number of events (see below). In outline, the algorithm proceeds as follows:

1. Make an initial assignment of the probabilities of each event belonging to each cluster. This can be based on the current classes, or can be done by randomly assigning events to clusters.
2. Using the probabilities of cluster membership, calculate the centres, shapes and number of events in each cluster.
3. Work out the probabilities of each event belonging to each cluster based on the new cluster centres and shapes.
4. Work out the most likely class code for each event. If any event is more than 3 times the Mahalanobis distance from the centre of the most likely class, set class 00.
5. If the most likely class assignment has not changed since the previous iteration or the iteration count is too high then stop, otherwise go back to step 2.

As set out, the algorithm converges slowly, so there are additional steps (not described here) required to accelerate the process. There is always the possibility that one or more classes will not be represented when iterating has finished. In this case, we take the class with the largest number of members and split this in two, then repeat this process until we have the desired number of classes, then we restart the iteration process.

There is no guarantee that any solution is the best as the outcome depends on the initial assignment of probabilities. If you choose to run the algorithm with no use of existing classifications, we repeat the process 10 times with randomised initial conditions and we choose the result that has the maximum probability density.

## The mathematical details

We want to partition our  $M$  events into the  $k$  most likely clusters. Given a partition, for each of the  $k$  clusters, we can work out a probability density. For the  $j$ th cluster we can write this as  $N_j * W_j$  where  $N_j$  is the normalised multivariate Normal distribution and  $W_j$  is the number of events in that cluster.

The normalised multivariate Normal distribution for a point at position  $\mathbf{x}$  relative to the centre of the distribution is given (in matrix notation) by:

$$N(\mathbf{x}) = \exp(-\frac{1}{2} \mathbf{x}' \mathbf{C} \mathbf{inv} \mathbf{x}) / ((2\pi)^n \det(\mathbf{C}))^{1/2}$$

where:

$n$  Is the number of dimensions of the distribution (in our case this is 2 or 3).

$\mathbf{x}$  Is a vector length  $n$  holding the position relative to the centre of the Normal distribution.

$\mathbf{C}$  Is a covariance matrix of size  $n$  by  $n$ . We construct the covariance from the positions of all the points in the cluster relative to the centre of the cluster.

$\mathbf{C} \mathbf{inv}$  Is the inverse of the covariance matrix.

$\det(\mathbf{C})$  Is the determinant of the covariance matrix.

We have  $M$  data points that we wish to cluster. We define the value  $G_i$  for the  $i$ th of our  $M$  points as:

$$G_i = \sum_j N_j(i) * W_j$$

where the sum for  $j$  is over the  $k$  clusters and  $N_j(i)$  is the normalised multivariate Normal distribution value at data point  $i$  relative to the centre of cluster  $j$ . This value  $G_i$  is the probability density for the  $i$ th point. We want to maximise the probability density for all points, which comes to maximising:

$$\prod_i G_i$$

where the product is over all  $M$  points. This number is likely to be inconveniently large, so we take its logarithm:

$$\ln(\prod_i G_i) = \sum_i \ln(G_i)$$

The value that is reported after the normal mixtures algorithm runs is this value divided by  $M$ , being the log probability density per point.

## Mahalanobis distance

When describing clustering and the ellipses drawn to illustrate class boundaries we have referred to the Mahalanobis distance (named after P.C. Mahalanobis who described this measure in the 1930s). Given multivariate data values for which the values in each variable are normally distributed around a mean, this measure allows us to define boundaries of constant probability around the multi-dimensional centre of the distribution.

In one dimension, the normal distribution leads to a probability density  $P(x)$  that is proportional to:

$$P(x) \propto \exp(-1/2x^2/v)$$

where  $x$  is the distance from the mean and  $v$  is the variance ( $\sigma^2$ ). In two and three dimensions, this becomes:

$$P(x, y) \propto \exp(-1/2(x^2/v_{xx} + 2xy/v_{xy} + y^2/v_{yy}))$$
$$P(x, y, z) \propto \exp(-1/2(x^2/v_{xx} + 2xy/v_{xy} + y^2/v_{yy} + 2yz/v_{yz} + z^2/v_{zz} + 2zx/v_{zx}))$$

where  $x$ ,  $y$  and  $z$  are the distances from the means and  $v_{ab}$  is the joint variance in the two components  $a$  and  $b$ . These equations generalise to any number of dimensions and are often written using matrix notation:

$$P(\mathbf{x}) \propto \exp(-1/2 \mathbf{x}' \mathbf{C} \mathbf{inv} \mathbf{x})$$

where  $\mathbf{x}$  is the vector of distances from the mean,  $\mathbf{x}'$  is the transpose of this, and  $\mathbf{C} \mathbf{inv}$  is the inverse of the covariance matrix.

The boundaries of constant probability for one, two and three dimensions satisfy the equations:

$$x^2 / \sigma^2 = r^2$$
$$x^2/v_{xx} + 2xy/v_{xy} + y^2/v_{yy} = r^2$$
$$x^2/v_{xx} + 2xy/v_{xy} + y^2/v_{yy} + 2yz/v_{yz} + z^2/v_{zz} + 2zx/v_{zx} = r^2$$

where  $r$  is a constant. By choosing suitable values of  $r$  we can set boundaries within which we would expect to find a certain proportion of the data. In one dimension, the boundary is a distance from the centre, in two dimensions it is an ellipse and in three dimensions this is an ellipsoid. When  $r$  is 1.0, we refer to the distance of constant probability density as the Mahalanobis distance. In the one-dimensional case, this is equivalent to the standard deviation.

# **18: Digital filtering**

# Digital filtering

## FIR and IIR filters

Filtering is used to remove unwanted frequency components from waveforms and can also be used to differentiate a signal. In Spike2 we provide you with two basic types of filter: Finite Impulse Response (FIR) and Infinite Impulse Response (IIR). Both types of filter have their advantages and disadvantages.

### IIR filters

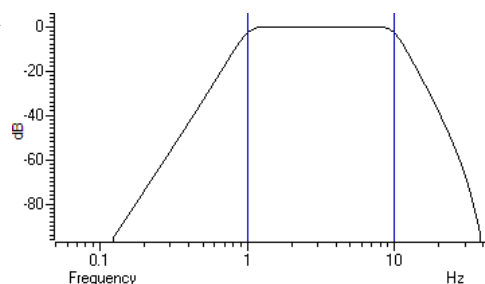
These are similar to analogue filters; we design them by mapping Butterworth, Bessel, Chebyshev filters and resonators into their digital forms. They have advantages:

- They generate much steeper edges and narrower notches than FIR filters for the same computational effort.
- They are causal; they do not use future data to calculate the output, so there is no pre-ringing due to transients.

They also have disadvantages:

- IIR filters are prone to stability problems particularly as the filter order increases or when a filter feature becomes very narrow compared to the sample rate.
- IIR filters impose a group delay on the data that varies with frequency. This means that they do not preserve the shape of a waveform, in particular, the positions of peaks and troughs will change.

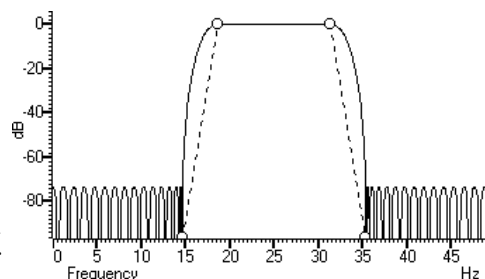
The output of an IIR filter may take a long time to settle down from the discontinuity at the start (transition from no data to the supplied data).



### FIR filters

We describe FIR filters in terms of frequency bands: *pass bands*, *stop bands* and *transition gaps*. You define a filter by the arrangement of bands and the corner frequencies of each band. FIR filters have these advantages:

- They are unconditionally stable as they do not feedback the output to the input.
- There is no phase delay through the filter, so peaks and troughs do not move when data is filtered (this is called *linear phase* in the literature).



They also have disadvantages:

- They are poor at generating very narrow notches or narrow band pass filters.
- The narrowest frequency band or band gap is limited by the number of coefficients (we allow up to 2047).
- FIR filters are not causal; they use future as well as past data to generate each output point. A transient in the input causes effects in the output before the transient.

If your FIR filter has  $n$  coefficients, the first and last  $n/2$  output points are estimates due to the discontinuity at the start and end of the data.

### So which to choose?

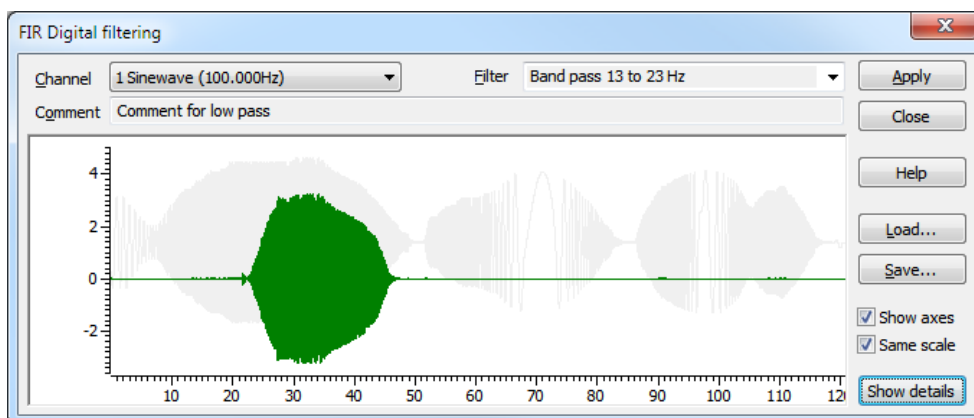
If you want a differentiating filter, you have no choice as we have not implemented IIR differentiators. Unless one of the FIR disadvantages is a problem, you will likely have fewer unexpected effects with an FIR filter.

However, there are circumstances in which only an IIR filter will do. If you need a high  $Q$  notch filter or resonator, then use an IIR filter. If you are interested in small changes just before a large discontinuity, only the IIR filter will help you. However, make sure that you understand the disadvantages of IIR filters before you depend on their output.



## Digital filter dialog

The Analysis menu **Digital filters** command is available when you have a data file open that contains waveform channels. You can choose to apply Finite Impulse Response (FIR) filters or Infinite Impulse Response (IIR) filters. Apart from the dialog title, the initial dialog display is the same for IIR and FIR filters. You can apply one of twelve stored digital filters to a waveform channel, or you can create your own digital filter. You can also load and save additional sets of filters from the dialog.



The dialog shows the original waveform in grey, and a filtered version in the colour you have set for waveform data in a time view. Whenever you change the filter, the display updates to show the effect of the change.

### Show axes

You can choose to **Show axes** for the original data and for the filtered version. The axis for the original data is always on the left. If a separate axis is required for the filtered data, it is drawn in the filtered data colour on the right.

### Same scale

Initially, the filtered data is drawn at the same scale as the original data. However, sometimes this is inconvenient, for example when high-pass filtering a signal with a significant DC offset or when the result of filtering is very small compared to the original. If you clear the **Same scale** check box, the filtered data is scaled to fit in the window independently of the scaling of the original waveform.

### Channel

The Channel field allows you to select a Waveform or a RealWave channel to filter.

### Filter

The Filter field of the dialog box selects the filter to apply. There are normally 12 filters to choose from. When you first open the dialog, this field is grey, indicating that you cannot edit the filter name. If you display the filter details you can modify the filter name.

### Comment

The Comment field is for any purpose you wish; there is one comment per filter. When you first open the dialog, this field is grey, indicating that you cannot edit the comment. Click the **Show details** button to edit the comment.

The Filter field of the dialog box selects the filter to apply and the Channel field sets the waveform channel to filter.

The **Close** button shuts the dialog and will ask if you want to save any changed filter and the **Help** button opens the on-line Help at the digital filtering topic.

### Close

Click the **Close** button to shut the filtering dialog. If you have made a change to any of the filters, or loaded a

new filter set, you are asked if you want to save the current set of filters as your standard filter set.

## Load and Save

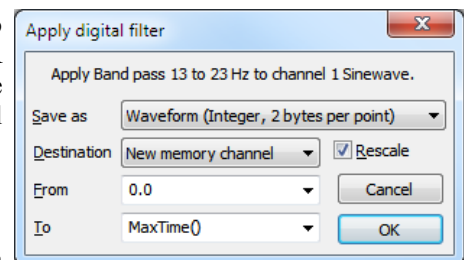
You can choose to save the current set of filters to a `.cfbx` (filter bank) file, or to load a new set of filters from a `.cfbx` or `.cfb` file. If you are working with FIR filters, this only saves or loads FIR filters. If you are working with IIR filters, this only saves or loads IIR filters. Loading a new filter set does not change your standard filter set, however, you will be asked if you want to change your standard set when you close the dialog.

## Show details

The **Show details** button increases the dialog size to display a new area in which you can design and edit filters. Click this button again to hide the new dialog area. When you display the filter details, the **Filter** and **Comment** fields become editable. If you change a filter or create a new filter or load a new set of filters from a file, you will be prompted to save the filter bank when you close the digital filter dialog. The details are different for FIR and IIR filters.

## Apply

The **Apply** button opens a new dialog in which you set where to write the result of the filter operation on the data channel. In the IIR filter dialog, this button is disabled if the filter is not valid. The channel comment of the new channel holds the source channel number and a description of the filtering operation.



## Save As

You can save the results of filtering as 16-bit integer data (Waveform) using a scale and offset to convert to user units, or as 32-bit floating point data (RealWave). RealWave output gives you a more accurate result at the cost of using twice as much storage space.

## Destination

You can write the result to a new memory channel, to an unused channel on disk, or to a memory channel that holds waveform or RealWave data with the same sampling rate as the channel you are filtering.

## From and To

The **From** and **To** fields set the time range to process.

## Rescale

If you check **Rescale**, the output data scale and offset are set to give the best possible representation of the waveform as 16-bit integers. If you save the output as a Waveform, this doubles the time required for filtering. If you **Rescale** when adding data to an existing memory channel, the scale and offset take into account both the added data and any remaining original data. If **Rescale** is unchecked, the scale and offset values from the source channel are copied. You would normally wish to rescale output to a RealWave channel as this improves accuracy if the channel is ever read as integer data.

## OK

Click **OK** to start filtering. As applying a filter can be a lengthy process, a progress dialog appears with a **Cancel** button during the filtering operation.

## Filter bank

A digital filter definition is complex and it would be tedious to specify all the properties of a filter each time you wanted to apply one to data. To avoid this, Spike2 contains a filter bank of 12 FIR and 12 IIR filter definitions. This filter bank is saved to the XML file `filtbank.cfbx` when you close Spike2 and reloaded

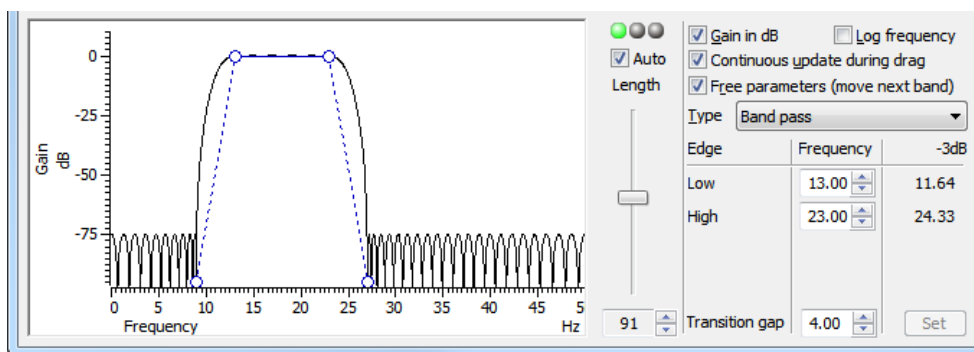
when you open it. If this file is not found, `filterbank.cfb` (the old format version) is read. If no filter bank file is found, Spike2 creates a new one with standard settings.

This default filter bank file is in the system folder for Spike2 user and application data, the same folder as returned by `FilePath$(-3)`. This is usually of the form: `C:\Users\username\AppData\Local\CED\Spike8\` where `username` is your user name. This means that each user of the system has their own default filter bank. You can load and save the filter bank files to other locations from the digital filter dialog.

When you use the digital filter dialog, you specify which filter you want by the filter name. Script users identify the filter by its type (FIR or IIR) and an index number in the range 0 to 11. Script users also have access to two additional temporary filters with index number -1 (one for FIR and the other for IIR filters) that they can set and use for channel filtering operations without changing the standard filter set.

## FIR filter details

The graph in the details area displays the ideal and actual frequency response of the filter. The ideal response appears as blue solid lines for each pass band linked by dotted lines that mark transition gaps between the bands. All transition gaps have the same width. The calculated frequency response is drawn as solid black lines and is greyed when the filter specification has changed and the response has not yet been calculated.



*Details section of the FIR filter dialog*

The mouse pointer changes to indicate the feature it is over:  $\leftarrow\rightarrow$  for a pass band or stop band,  $\leftarrow\circ\rightarrow$  for a transition gap and  $\leftarrow\circ$  for a band corner. The small circles can be dragged sideways to change the slope of the band edges or you can edit the band edges as numbers in the Frequency panel on the right. You can also drag the bands and the transition gaps sideways.

### Set

If you edit the numbers in the Frequency panel, the Set button is enabled so you can force a recalculation of the filter.

### -3dB point

The filters produced by the program are not defined in terms of -3dB corner frequencies and  $n$  dB per octave, as is often the case for traditional analogue filters. The 3dB point column is present to help users who are more comfortable describing filter band edges in terms of the 3 dB point.

### Gain in dB

The Gain in dB check box sets the y axis scale to dB if checked, linear if not checked. Using dB is usually the more convenient, except when working on a differentiator.

### Log frequency

You can choose to display the frequency axis as the logarithm of the frequency, which gives you more resolution at low frequencies. However, this can make working with FIR filters more awkward as it removes the visual symmetry of the transition bands. In log mode, the frequency axis extends down to 0.001 of the data sample rate.

### Continuous update

If you check the **Continuous update** box, the filter is updated while you drag the filter features around. If you have a slow computer and this feels ponderous you can clear the check box, in which case the filter is not recalculated until you stop changing features.

### Free parameters

If you check the **Free parameters** box, dragged features are not limited by the next band and will push bands along horizontally. If you clear the box, the horizontal motion of a dragged feature is limited by the next moveable object.

### Length, Auto and traffic lights

To the right of the frequency response display is a slider that controls the number of filter coefficients. In general, the more coefficients, the better the filter. However, the more coefficients, the longer it takes to compute them and the longer to filter the data. If you check the **Auto** box, the program will adjust the number of coefficients for you to produce a useful filter (with around 70 dB cut between a pass band and a stop band). The “traffic light” display above the slider shows green if the filter is good, amber if the result is usable but not ideal, and red if the result is hopeless.

An FIR filter of length  $n$  uses the  $n/2$  points before and after each input point to produce each output point. When there is no input data available before or after an input point, the filter uses a duplicate of the nearest point as an estimate of the data value. The  $n/2$  output points next to any break in the input data should not be used for any critical purpose.

## FIR Filter types

The **Type** field sets the arrangement of filter bands. If you need a filter that is not in this list you can generate it from the script language with the `FIRMake()` command. There are currently 12 different filter types:

### All pass

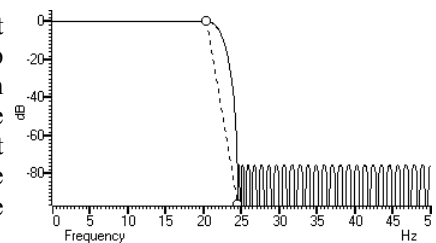
This has no effect on your signal. This filter type covers the case where you apply a low pass filter designed for a higher sampling rate to a waveform with a much lower sampling rate, so that the pass band extends beyond half the sampling frequency of the new file.

### All stop

This removes any signal; the output is always zero. This filter type is provided to cover the case where you apply a high pass filter designed for a higher sampling rate to a waveform with a much lower sampling rate, so that the stop band extends beyond half the sampling frequency of the new file.

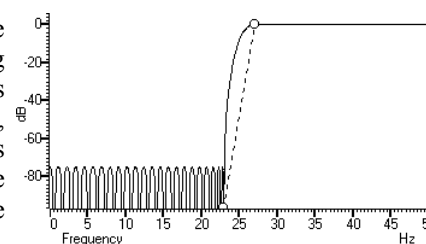
### Low pass

This filter attempts to remove the high frequencies from the input signal. The filter has three bands: a pass band starting at 0 Hz, a stop band ending at half the sample rate and a transition band between them. The **Frequency** field holds one editable number, **Low pass**, the frequency of the upper edge of the pass band. The stop band starts at this frequency plus the value set by the **Transition gap** field. The number of coefficients needed to realise the filter is determined by the narrowest of the bands.



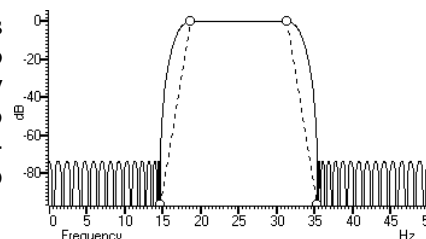
## High pass

A high pass filter removes low frequencies from the input signal. The filter has three bands: a stop band starting at 0 Hz, a pass band ending at half the sample rate and a transition band between the stop and pass bands. The **Frequency** field holds one editable number, **High pass**, the frequency of the lower edge of the pass band. The stop band starts at this frequency less the value set by the **Transition gap** field. The narrowest band determines the number of coefficients needed to realise the filter.



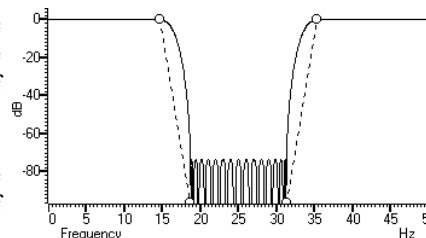
## Band pass

A band pass filter passes a range of frequencies and removes frequencies above and below this range. The filter has five bands: two stop bands, two transition bands and one pass band. The **Frequency** field has two editable numbers, **Low** and **High**, which correspond to the two edges of the pass band. The stop band below runs up to **Low-Transition gap**, and the stop band above from **High+Transition gap** to one half the sampling rate.



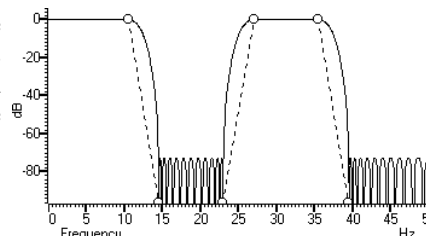
## Band stop

A band stop filter removes a range of frequencies. The filter has five bands: two pass bands, two transition bands and one stop band. The **Frequency** field has two editable numbers, **High** (the upper edge of the first pass band) and **Low** (the lower edge of the upper pass band). The stop band between the two pass bands runs from **High+Transition gap** up to **Low-Transition gap**. If you are trying to remove a single frequency you should consider using an IIR resonator notch filter. If you are trying to remove mains pickup, see the CED web site and search for "hum remove".



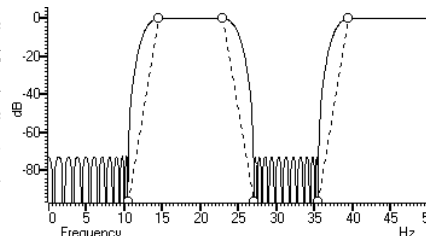
## One and a half low pass

This filter has two pass bands, the first running from zero Hz and the second in the frequency space between the upper edge of the first pass band and one half the sampling rate. There are also two stop bands and three transition bands. The **Frequency** field has three editable numbers: **Band 1 high**, **Band 2 low** and **Band 2 high**. These numbers correspond to the edges of the pass bands. You are unlikely to need this filter in standard applications.



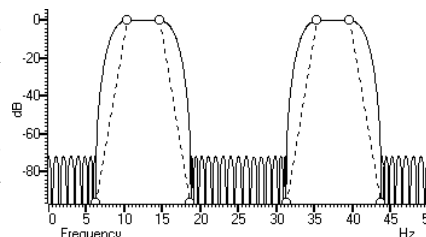
## One and a half high pass

This filter has two pass bands. The second runs up to one half the sampling rate. The first band lies in the frequency space between 0 Hz and the lower edge of the second band. It also has two stop bands and three transition bands. The **Frequency** field has three editable numbers: **Band 1 low**, **Band 1 high** and **Band 2 low**. These numbers correspond to the edges of the pass bands. You are unlikely to need this filter in standard applications.



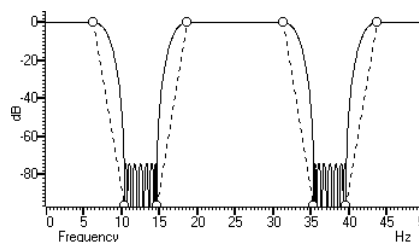
## Two band pass

This filter passes two frequency ranges and rejects the remainder. It is implemented by three stop bands, two pass bands and four transition bands. Both 0 Hz and one half the sampling frequency are rejected. The **Frequency** field has 4 numeric fields: **Band 1 low**, **Band 1 high**, **Band 2 low** and **Band 2 high**. These fields correspond to the four edges of the two pass bands. You are unlikely to need this filter in standard applications.



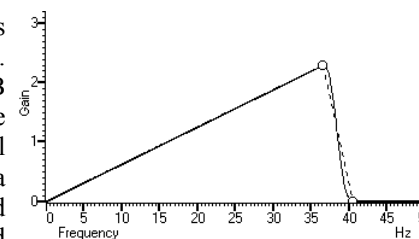
### Two band stop

This filter passes three frequency ranges and rejects the remainder. It is implemented by three stop bands, two pass bands and four transition bands. Both 0 Hz and one half the sampling rate are passed. The Frequency field has 4 numeric fields: Band 1 high, Band 2 low, Band 2 high and Band 3 low. These fields correspond to the four edges of the three bands. You are unlikely to need this filter in standard applications.



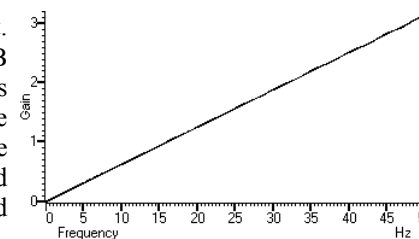
### Low pass differentiator

This filter is a combination of a differentiator (that is the output is proportional to the rate of change of the input) and a low pass filter. The y axis scale in the image to the right is linear, rather than in dB (although you can display it in dB if you wish). There is one editable number in the Frequency field, Low pass, the end of the differential section of the filter. The filter is implemented as three bands: a differentiation band starting at 0, a transition band and a stop band running up to half the sampling rate. The number of coefficients needed to realise the filter depends on the narrowest band.



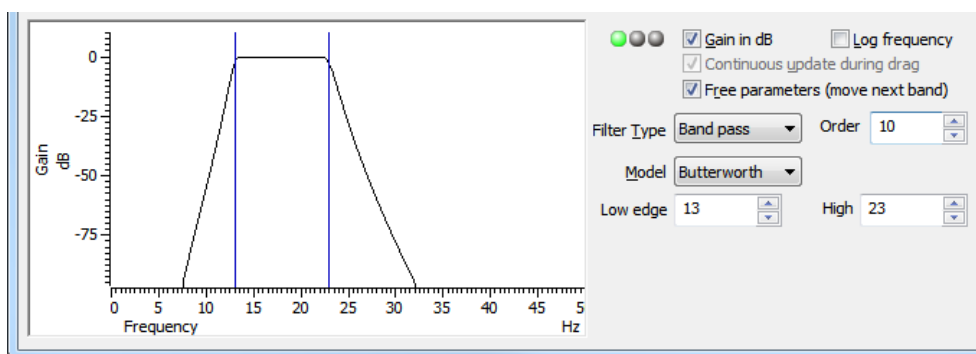
### Differentiator

The output of the filter is proportional to the rate of change of the input. The y axis scale in the image to the right is linear, rather than in dB (although you can display it in dB if you wish). The Frequency field is empty as there is only one band and it extends from 0 Hz to half the sampling rate. The number of coefficients determines the ripple (the deviation from the ideal of a linear relationship between frequency and gain). This type of filter must have an even number of coefficients, and the output is displaced 0.5 of a sample from the input.



## IIR filter details

We describe IIR filters in terms of a filter type (low pass, high pass, band pass or band stop), the analogue filter model that they are based on (Butterworth, for example), the corner frequencies and the filter order (which determines the steepness of the cut-off outside the desired pass bands). Filters based on Chebyshev designs also require a ripple specification and resonators require a Q factor.



Details section of the IIR filter dialog

The graph in the details area displays the corner frequencies and the frequency response of the filter. You can adjust the filter by clicking and dragging in this area or by editing the filter parameters as text. The mouse pointer changes to indicate the feature it is over:  $\leftarrow \rightarrow$  for a corner frequency and  $\updownarrow$  for an adjustable parameter (ripple or Q factor).

### Gain in dB

The Gain in dB check box sets the y axis scale to dB if checked, linear if not checked. Using dB is usually the

more convenient.

### Log frequency

You can choose to display the frequency axis as the logarithm of the frequency, which gives you more resolution at low frequencies. The display extends to 0.0001 of the sample rate. If you need a corner frequency below this you should consider sampling the signal more slowly in the first place. Alternatively, low pass filter the signal and then use a channel process to down sample it before filtering.

### Continuous update

This is always checked for IIR filters, and is greyed out.

### Free parameters

If you check the Free parameters box, corner frequencies are not limited by the next corner, and will push them along. If you clear the box, corner frequencies cannot be moved past each other.

### Traffic lights and messages

The traffic lights show green if the filter appears to be OK, amber if the filter may be unstable and red if the filter calculation failed, the filter is unstable or if one of the input parameters is illegal. There will usually be an explanatory message in the lower right hand corner of the dialog explaining the problem. In the amber state, the filter may still be usable; you can look at the frequency response and the filtered data to see if the result is acceptable.

### Filter Order

In an analogue filter, the filter order determines the sharpness of the filter, for example Butterworth and Bessel filters tend towards 6n dB per octave, where n is the filter order. In a digital filter, the order determines the number of filter coefficients. The higher the order, the sharper the filter cut-off and also, the more likely the filter is to be unstable due to problems in numerical precision. You can set filter orders of 1 to 10. You should always use the lowest order that meets your filtering criteria. Phase non-linearity gets worse as the filter order increases.

### Filter type

There are four filter types: Low pass, High pass, Band pass and Band stop. However, if you set the filter model to Resonator, there are only two types, being Band pass (this creates a resonator filter) and Band stop (this creates a notch filter). For all filter models except Chebyshev type 2 and Resonator, the frequencies given are the points at which the filter achieves a cut of 3 dB. For Chebyshev type 2 filters, the frequencies are the point at which the filter cut reaches the value set by the Ripple parameter. For Resonators, the frequency is the centre frequency of the resonator.

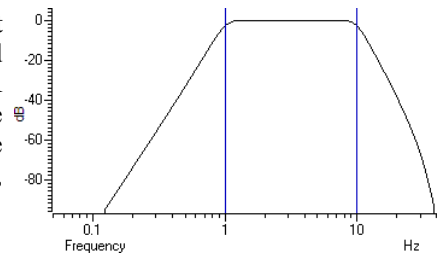
|           |                                                                                                                                                        |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| Low pass  | Use this to remove high frequencies and pass low frequencies. This has a single corner frequency.                                                      |
| High pass | Use this to remove low frequencies and pass high frequencies. This has a single corner frequency.                                                      |
| Band pass | This passes a range of frequencies between the Low edge and the high edge. If the filter model is Resonator, then this has a single Centre frequency.  |
| Band stop | This removes a range of frequencies between the Low edge and the high edge. If the filter model is Resonator, then this has a single Centre frequency. |

### Filter Model

The IIR filters we provide are digital implementation of standard analogue filter models. The following descriptions use fifth order 1 to 10 Hz band pass filters on 100 Hz data as examples (except for the Resonator filters). The five filter models are:

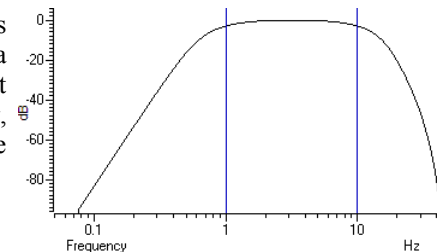
**Butterworth**

The Butterworth filter has a maximally flat pass band, but pays for it by not having the steepest possible transition between the pass band and the stop band. This is a good choice for a general-purpose IIR filter, but beware that the group delay can get quite bad near the corners, especially for high-order filters. Put another way, the shape of a signal can be significantly distorted. If signal shape is important, you could consider using a Bessel filter.



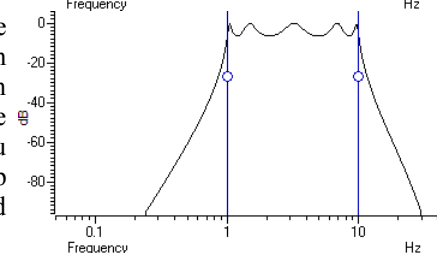
**Bessel**

An analogue Bessel filter has the property that the group delay is maximally flat, which means that it tends to preserve the shape of a signal passed through it. The price you pay for this property is that it leads to filters with a gentle cut-off. When realised as a digital filter, the constant group delay property is compromised; the higher the filter order, the worse the group delay.



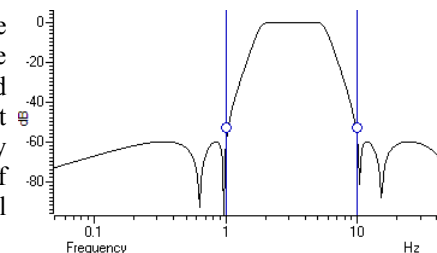
**Chebyshev type 1**

Filters of this type are based on Chebyshev polynomials and have the fastest transition between the pass band and the stop band for a given ripple in the pass band and no ripples in the stop band. You can adjust the ripple by dragging the small circles vertically or with the Ripple field to the right of the displayed frequency response. You will likely select this type of filter when having no signal in the stop band is more important to you than the flatness of the pass band and you do not care about preserving the signal shape.



**Chebyshev type 2**

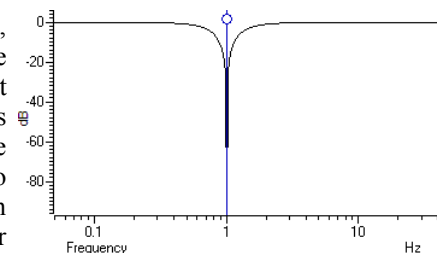
Filters of this type are defined by the start of the stop band and the stop band ripple (the minimum cut in the stop band). They have the fastest transition between the pass and stop bands given the stop band ripple and no ripple in the pass band. Drag the small circles to adjust the ripple, or use the Ripple field to the right of the frequency response. These filters can be useful when you need a sharp cut off and a flat pass band and do not care about preserving the signal shape.



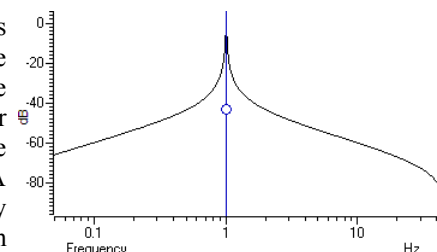
**Resonator**

Resonators are defined by a centre frequency and a Q factor. The Q is the centre frequency divided by the width of the resonator at the -3 dB point. You can have a band stop or a band pass resonator. The Q is adjusted by dragging the small circle or with the Q field to the right of the displayed frequency response.

Band stop resonators are also called Notch filters. The higher the Q, the narrower the notch. Notch filters are sometimes used to remove mains hum, but if you do this you will likely need to set notches at the first few odd harmonics of the mains frequency and notch filters are not recommended if you care about the phase response of the signal in the vicinity of the notch. There are much better ways to remove hum than this, see for example the scripts for this purpose on our web site. The example has a very low Q (1.24) to make the filter response visible.



A band pass resonator is the inverse of a notch. Band pass resonators are sometimes used as alternatives to a narrow bandpass filter. The example has a Q of 100. The higher the Q set for a resonator, the longer it will take for the output to stabilise at the start of the filter output. Resonators, like Notches, are quite unpleasant filters in the way that they distort signal frequencies around central frequency. A resonator can be useful to detect the presence of a frequency component, but it is unlikely that the shape of the filtered waveform will have any meaning.



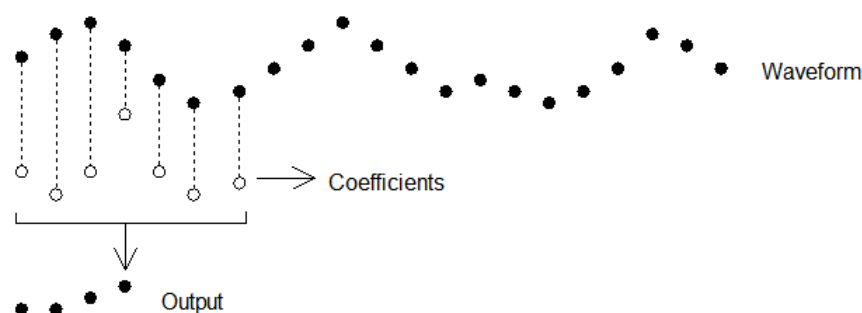


## FIR filters and scripts

The `FIRMake()`, `FIRQuick()` and `FiltCalc()` script commands and the Analysis menu **Digital filters...** dialog generate FIR (Finite Impulse Response) filter coefficients suitable for a variety of filtering applications. The generated filters are optimal in the sense that they have the minimum ripple in each defined band. These filter coefficients are used to modify a sampled waveform, usually to remove unwanted frequency components. The algorithmic heart of the filter coefficient generation is based on the well-known FORTRAN program written by Jim McClellan of Rice University in 1973 that implements the *Remez exchange algorithm* to optimise the filter.

The theory of FIR filters is beyond the scope of this document. Readers who are interested in learning more about the subject should consult a suitable text book, for example *Theory and Application of Digital Signal Processing* by Rabiner and Gold published by Prentice-Hall, ISBN 0-13-914101.

## FIR filtering algorithm



This diagram shows the general principle of the FIR filter. The hollow circles represent the filter coefficients, and the solid circles are the input and output waveforms. Each output point is generated by multiplying the waveform by the coefficients and summing the result. The coefficients are then moved one step to the right and the process repeats.

From this description, you can see that the filter coefficients (from right to left) are the *impulse response* of the filter. The impulse response is the output of a filter when the input signal is all zero except for one sample of unit amplitude. In the example above with 7 coefficients, there is no time shift caused by the filter. With an even number of coefficients, there is a time shift in the output of half a sample period.

## Frequencies

The Analysis menu **Digital filters...** command deals with frequencies in Hz as this is comfortable for us to work with. However, if you calculate a FIR filter for one sampling rate, and apply the same coefficients to a waveform sampled at another rate, all the frequency properties of the filter are scaled by the relative sampling rates. That is, the frequency properties of an FIR filter are invariant when expressed as fractions of the sampling rate, not when expressed in Hz.

It is usually more convenient when dealing with real signals to describe filters in terms of Hz, but this means that each time a filter is applied to a waveform the sampling rate must be checked. If the rate is different from the rate for which the filter was last used, the coefficients must be recalculated. Unless you use the `FIRMake()` script command, Spike2 takes care of all the frequency scaling and recalculation for you. The remainder of this description is to help users of the `FIRMake()` script command, but the general principles apply to all the digital filtering commands in Spike2.

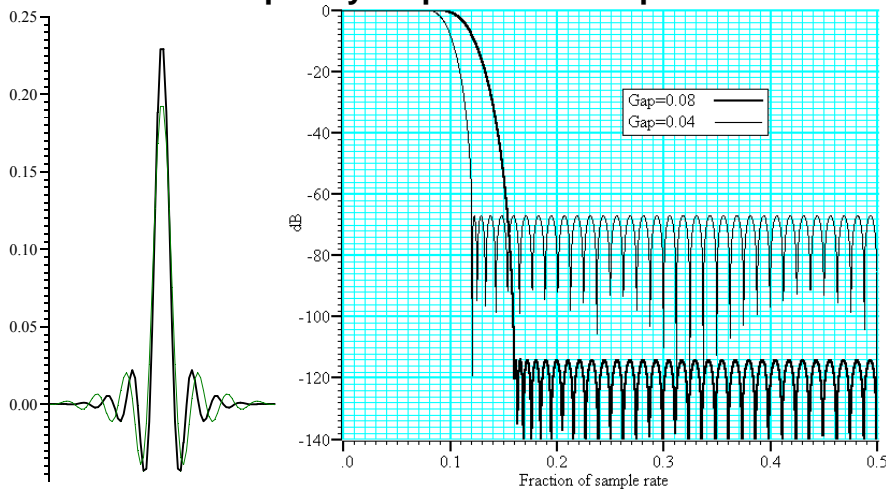
Users of the `FIRMake()` script command must specify frequencies in terms of fractions of the sample rate from 0 to 0.5. For example, if you were sampling at 10 kHz and you wanted to refer to a frequency of 500 Hz, you would call this 500/10000 or 0.05.

## Example filter

The heavy lines in the next diagrams show the results obtained by `FIRMake()` when it designed a low pass filter with 80 coefficients with the specification that the frequency band from 0 to 0.08 should have no attenuation, and that the band from 0.16 to 0.5 should be removed. We can specify the relative weight to give to the ripple in each band. In this case, we said that it was 10 times more important that the *stop band* (0.16 to 0.5) should pass no signal than the *pass band* should be completely flat.

We have shown the coefficients as a waveform for interest as well as the frequency response of the filter. The shape shown below is typical for a band pass filter. One way of understanding the action of the FIR filter is to think of the output as the correlation of the waveform and the filter coefficients.

### Coefficients and frequency response for low pass filters



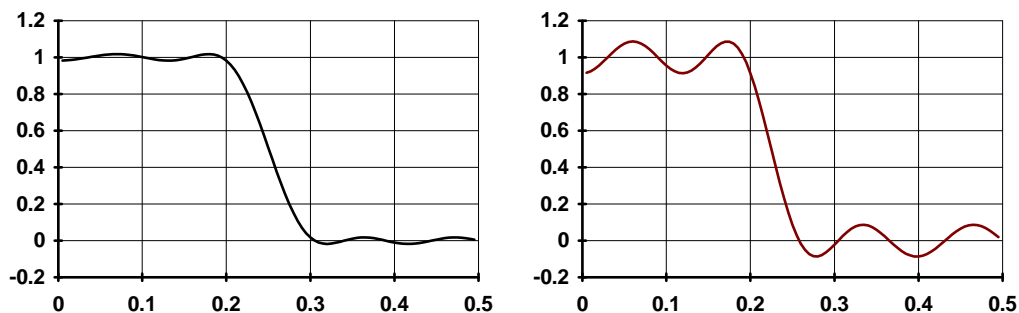
The frequency response is shown in dB, which is a logarithmic scale. A ratio  $r$  is represented by  $20 \log_{10}(r)$  dB. A change of 20 dB is a factor of 10 in amplitude, 6 dB is approximately a factor of 2 in amplitude. The graph shows that a frequency in the stop band is attenuated by over 110 dB (a factor of 300,000 in amplitude with respect to the signal before it was filtered).

Because we didn't specify what happened between a frequency of 0.08 and 0.16 of the sampling rate, the optimisation pays no attention to this region. You might ask what happens if we make this transition gap smaller. The lighter line in the graph shows the result of halving the width of the gap by making the stop band run from 0.12 to 0.5. The filter is now much sharper. However, you don't get something for nothing. The attenuation in the stop band is reduced from 110 dB to around 70 dB. Although you cannot see it from the graph, the ripple in the pass band also increases by the same proportion (from 1 part in 30,000 to 1 part in 300).

We can restore the attenuation in the stop band by increasing the number of coefficients to around 120. However, there are limits to the number of coefficients it is worth having (apart from increasing the time it takes to calculate the filter and filter the data). Although the process used to calculate coefficients uses double precision floating point numbers, there are rounding errors and the larger the number of coefficients, the larger the numerical noise in the result.

Because the waveform channels are stored in 16-bit integers, there is no point designing filters that attenuate any more than 96 dB as this is a factor of 32768. Attenuations greater than this would reduce any input to less than 1 bit. If you are targeting data stored in real numbers this restriction may not apply.

It is important that you leave gaps between your bands. The smaller the gap, the larger the ripple in the bands.



This is illustrated by these two graphs. They show the linear frequency response of two low pass filters, both designed with 18 coefficients (we have used so few coefficients so the ripple is obvious). Both have a pass band of 0 to 0.2, but the first has a gap between the pass band and the stop band of 0.1 and the second has a gap of 0.05. We have also given equal weighting to both the pass and the stop bands, so you can see that the ripple around the desired value is the same for each band.

As you can see, halving the gap has made a considerable increase in the ripple in both the pass band and the stop band. In the first case, the ripple is 1.76%, in the second it is 8.7%. Halving the transition region width increased the ripple by a factor of 5.

In case you were worrying about the negative amplitudes in the graphs, a negative amplitude means that a sine-wave input at that frequency would be inverted by the filter. The graphs with dB axes consider only the magnitude of the signals, not the sign.

## FIRMake filter types

`FIRMake()` can generate coefficients for four types of filter: Multiband, Differentiators, Hilbert transformer and a variation on multiband with 3 dB per octave frequency cut. The other routines can generate only Multiband filters and Differentiators.

### Multiband filters

The filter required is defined in terms of frequency bands and the desired frequency response in each band (usually 1.0 or 0.0). Bands with a response of 1.0 are called *pass bands*, bands with a response of 0.0 are called *stop bands*. You can also set bands with intermediate responses, but this is unusual. The bands may not overlap, and there are gaps between the defined bands where the frequency response is undefined. You give a weighting to each band to specify how important it is that the band meets the specification. As a rule of thumb, you should make the weight in stop bands about ten times the weight in pass bands.

`FIRMake()` optimises the filter by making the ripple in each band times the weight for the band the same. The ripple is the maximum error between the desired and actual filter response in a band. The ripple is usually expressed in dB relative to the unfiltered signal. Thus the ripple in a stop band is the minimum attenuation found in that band. The ripple in a pass band is the variation of the frequency response from the desired response of unity. In some situations, for example audio filters, quite large ripples in the pass band are tolerable but the same ripple would be unacceptable in a stop band. For example, a ripple of -40 dB in a pass band (1%) is inaudible, but the same ripple in a stop band would allow easily audible signals to pass. By weighting bands you can increase the attenuation in one band at the expense of another to suit your application.

### Differentiators

The output of a differentiator increases linearly with frequency and is zero at a frequency of 0. The differentiator is defined in terms of a frequency band and a slope. The frequency response at frequency  $f$  is  $f * slope$ . The slope is usually set so that the frequency response at the highest frequency is no more than 1.

The weight given to each frequency within a band is the weight for that band divided by the frequency. This gives a more accurate frequency response at low frequencies where the resultant amplitude will be the smallest.

Although you can define multiple bands for a differentiator, it is unusual to do so. Almost all differentiators define a single band that starts at 0. Occasionally a differentiator followed by a stop band is needed.

## Hilbert transformers

A Hilbert transformer is a very specialised form of filter that causes a phase shift of  $-\pi/2$  in a band, often used to separate a signal from a carrier. The theory and use of this form of filter is way beyond the scope of this document. Unless you know that you need this filter type you can ignore it.

## Multiband with 3dB octave cut

This is a variation on the multiband filter that can be used to filter white noise to produce band limited pink noise. The filter is identical to the band pass filter except that the attenuation increases by 3 dB per octave in the band (each doubling of frequency reduces the amplitude of the signal by a factor of the square root of 2). It is used in exactly the same way as the multiband filter.

## Low pass filter example

A waveform is sampled at 1 kHz and we are interested only in frequencies below 100 Hz. We would like all frequencies above 150 Hz attenuated by at least 70 dB.

A low pass filter has two bands. The first band starts at 0 and ends at 100 Hz, the second band starts at 150 Hz and ends at half the sampling rate. Translated into fractions of the sampling rate, the two bands are 0-0.1 and 0.15 to 0.5. The first band has a gain of 1, the second band has a gain of 0. We will follow our own advice and give the stop band a weight of 10 and the pass band a weight of 1. We will try 40 coefficients to start with, so a possible script is:

```
var prm[5][2]; 'Array for parameters
var coef[40]; 'Array for the coefficients
' band start band end function weight
prm[0][0]:=0.00; prm[1][0]:=0.1; prm[2][0]:=1.0; prm[3][0]:= 1.0;
prm[0][1]:=0.15; prm[1][1]:=0.5; prm[2][1]:=0.0; prm[3][1]:=10.0;
FIRMake(1, prm[1][], coef[]);
PrintLog("Pass Band ripple=%.1fdB Stop band attenuation=%.1f\n",
 prm[4][0], prm[4][1]);
```

If you run this, the log view output is:

```
Pass Band ripple=-28.8dB Stop band attenuation=-48.8
```

The attenuation in the stop band is only 48 dB, which is not enough. The ripple in the pass band is around 3% of the signal amplitude. We can increase the stop band attenuation in three ways: by increasing the number of coefficients, by giving the stop band more weight, or by making the gap larger between the bands.

We don't want to give the stop band more weight; this would increase the ripple in the pass band. We could probably reduce the width of the pass band a little as the attenuation of the signal tends to start slowly, but we will leave that adjustment to the end. The best way to improve the filter is to increase the number of coefficients. If we increase the size of `coef[]` to 80 coefficients and run again, the output now is:

```
Pass Band ripple=-58.7dB Stop band attenuation=-78.7
```

This is much closer to the filter we wanted. You might wonder if there is a formula that can predict the number of coefficients based on the filter specification. There is no exact relationship, but the following formula, worked out empirically by curve fitting, predicts the number of coefficients required to generate a filter with equal weighting in each of the bands and is usually accurate to within a couple of coefficients. The formula can be applied when there are more than two bands, but becomes less accurate as the number of bands increase.

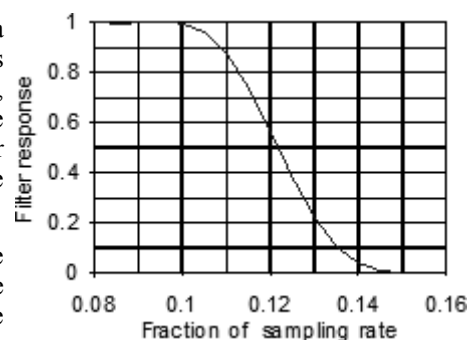
```
' dB is the mean ripple/attenuation in dB of the bands
' deltaF is the width of the transition region between the bands
' return An estimate of the number of coefficients
Func NCoefMultiBand(dB, deltaF)
return (dB-23.9*deltaF-5.585)/(14.41*deltaF+0.0723);
end;
```

In our example we wanted at least 70 dB attenuation, and we weighted the stop band by a factor of 10 (20 dB). This causes a 10 dB improvement in the stop band at the expense of a 10 dB degradation of the pass band. Thus to achieve 70 dB in the stop band with the weighting, we need 60 dB without it. If we set these values in the formula ( $dB = 60$ ,  $\delta F = 0.05$ ), it predicts that 67.13 coefficients are needed. If we run our script with 67 coefficients, we get 70.9 dB attenuation, which is close enough!

### A final finesse

If we look at the frequency response of our filter in the area between the pass band and the stop band, we see that the curve is quite gentle to start with. If you are used to using analogue filters, you will recall that the corner frequency for a low pass analogue filter is usually stated to be the frequency at which the filter response fell by 3 dB which is a factor of  $\sqrt{2}$  in amplitude (when the response falls to 0.707 of the unfiltered amplitude).

If we use the analogue filter definition of corner frequency, we see that we have produced a filter that passes from 0 to 0.115 of the sampling rate, and we wanted from 0 to 0.1, so we can move the corner frequency back. This will increase the attenuation in the stop band, and reduce the filter ripple, as it widens the gap between the pass band and the stop band. If we move it back to 0.085, the attenuation in the stop band increases to 84 dB. Alternatively, we could move both edges back, keeping the width of the gap constant. This leaves the stop band attenuation more or less unchanged, but means that the start of the stop band is moved lower in frequency.

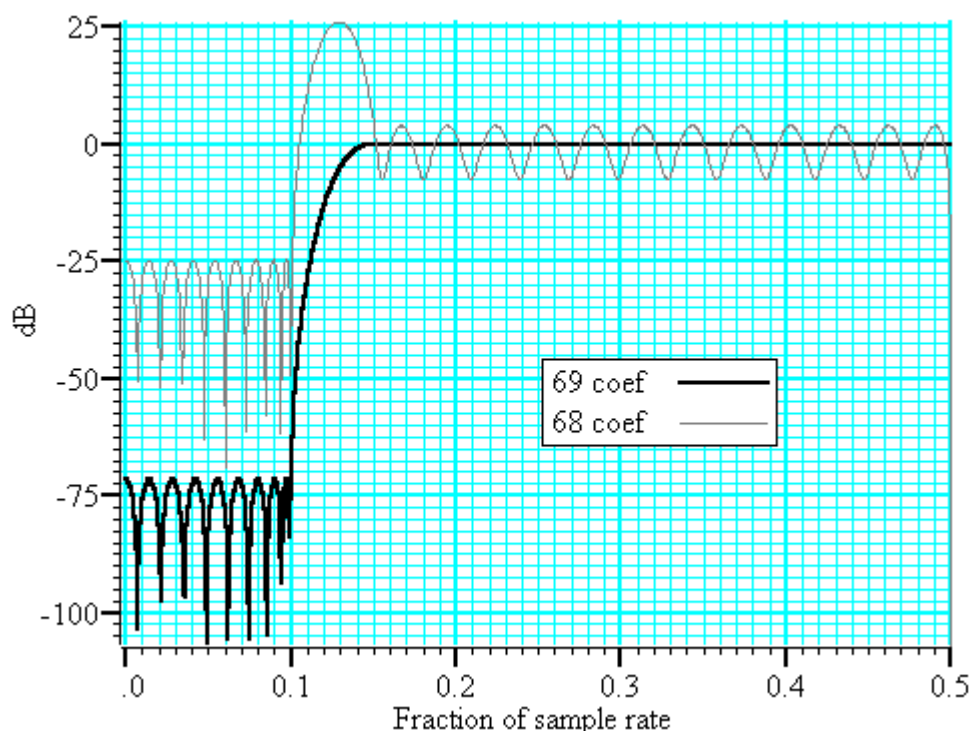


## High pass filter

A high pass filter is the same idea as a low pass except that the first frequency band is a stop band and the second band is a pass band. All the discussion for a low pass filter applies, with the addition that **there must be an odd number of coefficients**. If you try to use an even number your filter will be very poor indeed. The example below shows a script for a high pass filter with the same bands and tolerances as for the low pass filter. We have added a little more code to draw the frequency response in a result view.

```
var prm[5][2];
var coef[69];
' band start band end function weight
prm[0][0]:=0.00; prm[1][0]:=0.1; prm[2][0]:=0.0; prm[3][0]:=10.0;
prm[0][1]:=0.15; prm[1][1]:=0.5; prm[2][1]:=1.0; prm[3][1]:= 1.0;
FIRMake(1, prm[[]], coef[]);
const bins% := 1000;
var fr[bins%];
FIRResponse(fr[], coef[], 0);
SetResult(bins%, 0.5/(bins%-1), 0, "Fr Resp", "Fr", "dB");
ArrConst([], fr[]);
Optimise(0);
WindowVisible(1);
```

### Effect of odd and even coefficients



The graph shows the results of this high pass filter design with 69 coefficients, which gives a good result, and with 68 coefficients, which does not. In fact, if we had not given a factor of 10 weight (20 dB) to the stop band, the filter with 68 coefficients would not have achieved any cut in the stop band at all!

The reason for this unexpected result is that we have specified a non-zero response at the Nyquist frequency (half the sampling rate). If you imagine a sine wave with a frequency of half the sample rate, each cycle will contribute two samples. The samples will be  $180^\circ$  out of phase, so if one sample has amplitude  $a$ , the next will have amplitude  $-a$ , the next  $a$  and so on. The filter coefficients are mirror symmetrical about the centre point for a band pass filter, so with an even number of coefficients, the result when the input waveform is  $a, -a, a, -a, \dots$  is 0. Another way of looking at this is to consider that a filter with an even number of coefficients produces half a sample delay. The output halfway between points that are alternately  $+a$  and  $-a$  must be 0.

You can use the formula given for the low pass filter to estimate the number of coefficients required, but you must round the result up to the next odd number.

## General multiband filter

You can define up to 10 bands. However, it is unusual to need more than three. The most common cases with three bands are called band pass and band stop filters. In a band pass filter, you set a range of frequencies in which you want the signal passed unchanged, and set the frequency region below and above the band to pass zero. In a band stop filter you define a range to pass zero, and set the frequency ranges above and below to pass 1.

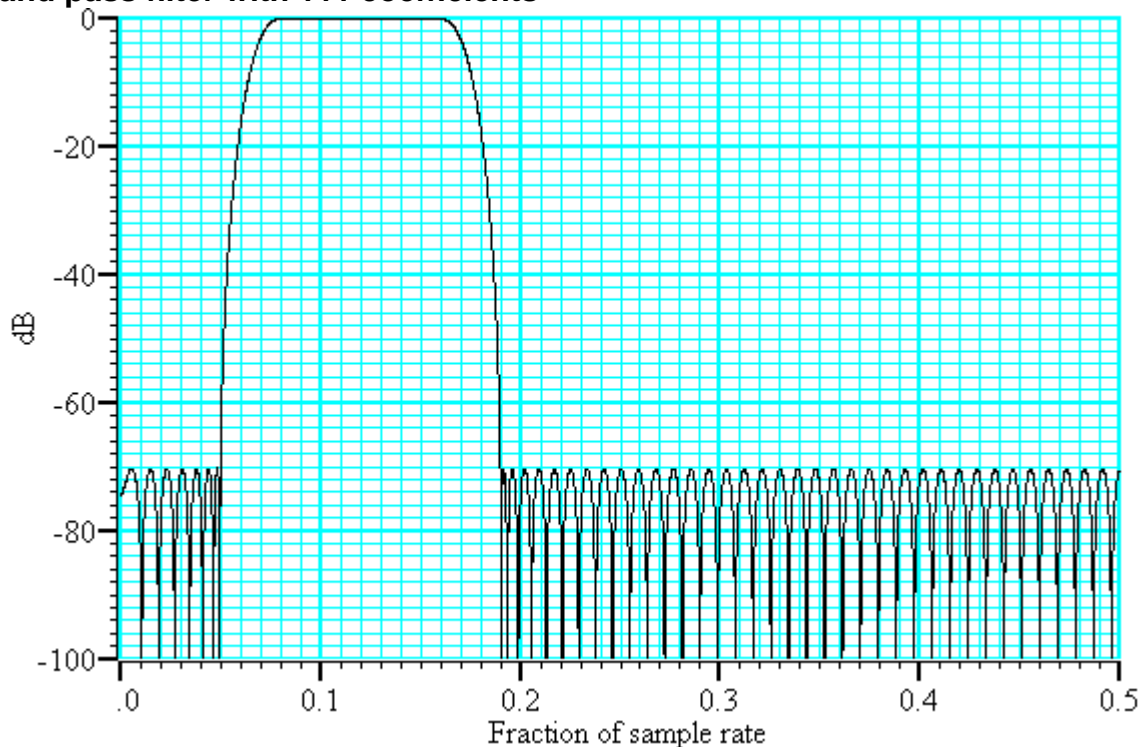
You must still allow transition bands between the defined bands, exactly as for the low and high pass filters, the only difference is that now you need two transition bands, not one. Also, if you want a non-zero response at the Nyquist frequency, you must have an odd number of coefficients.

For our example we will take the case of a signal sampled at 250 Hz. We want a filter that passes from 20 to 40 Hz (0.08 to 0.16) with transition regions of 7.5 Hz (0.03). If we say it is 10 times more important to have no signal in the stop band than ripple in the pass band, and we want 70 dB cut in the stop band we will get 50 dB ripple in the pass band (because a factor of 10 is 20 dB). To use the formula for the number of coefficients we need the mean attenuation/ripple in dB and the width of the transition region. The two stop bands have an attenuation of 70 dB and the pass band has a ripple of 50 dB, so the mean value is  $(70+50+70)/3$  or 63.33 dB. We have two transition regions (both the same width). In the general case of transition regions of different sizes, use the smallest transition region in the formula. Plugging these values into the formula predicts 113

coefficients, however only 111 are needed to achieve 70 dB.

```
var prm[5][3]; ' 3 bands for band pass
var coef[111]; ' 111 coefficients needed
' band start band end function weight
prm[0][0]:=0.00; prm[1][0]:=0.05; prm[2][0]:=0.0; prm[3][0]:=10.0;
prm[0][1]:=0.08; prm[1][1]:=0.16; prm[2][1]:=1.0; prm[3][1]:= 1.0;
prm[0][2]:=0.19; prm[1][2]:=0.50; prm[2][2]:=0.0; prm[3][2]:=10.0;
FIRMake(1, prm[0][], coef[0]);
```

### Band pass filter with 111 coefficients



## Differentiators

A differentiator filter has a gain that increases linearly with frequency over the frequency band for which it is defined. There is also a phase change of 90 degrees ( $\pi/2$ ) between the input and the output.

### Ideal differentiator with slope of 2.0

You define the differentiator by the number of coefficients, the frequency range of the band to differentiate and the slope. The example above has a slope of 2. Within each band (normally only 1 band is set) the program optimises the filter so that the amplitude of the ripple (error) is proportional to the response amplitude. A differentiator is normally defined to operate over a frequency band from zero up to some frequency  $f$ . If  $f$  is 0.5, or close to it, you must use an even number of coefficients, or the result is very poor. You can estimate the number of coefficients required with the following function:

```
' dB the proportional ripple expressed in dB
' f the highest frequency set in the band
' even% Non-zero if you want an even number of coefficients
func NCoefDiff(dB, f, even%)
if (f<0) or (f>0.5) then return 0 endif;
f := 0.5-f;
var n%;
if (even%) then
 n% := (dB+43.837*f-35.547)/(0.22495+29.312*f);
 n% := (n%+1) band -2; 'next even number
else
 if f=0.0 then return 0 endif;
 n% := dB/(29.33*f);
 n% := n% bor 1; 'next odd number
```

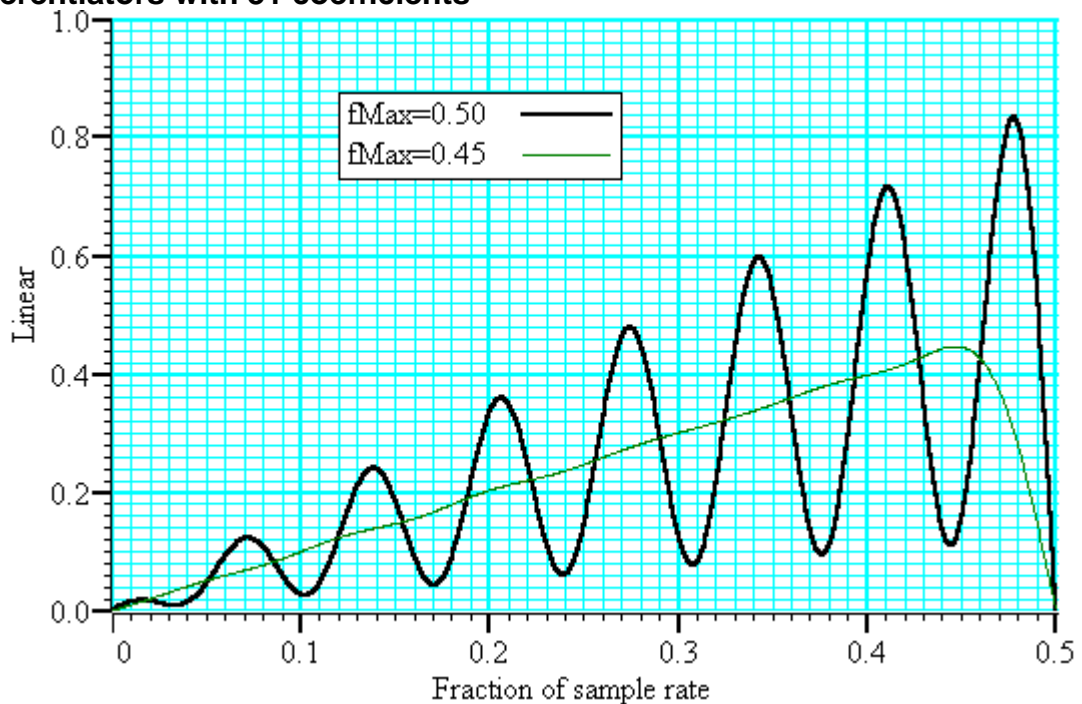
```
endif;
return n%;
end
```

For an even number of coefficients this is unreliable when  $f$  is close to 0.5. For an odd number, no value of  $n$  works if  $f$  is close to 0.5.

These equations were obtained by curve fitting and should only be used as a guide. To make a differentiator that uses a small number of coefficients, use an even number of coefficients and don't try to span the entire frequency range. If you cannot tolerate the half point shift produced by using an even number of coefficients and must use an odd number, you must set a band that stops short of the 0.5 point. Remember, that by not specifying the remainder of the band you have no control over the effect of the filter in the unspecified region. However, for an odd number of points, the gain at the 0.5 point will be 0 whatever you specify for the frequency band.

The graph below shows the effect of setting an odd number of coefficients when generating a differentiator that spans the full frequency range. The second curve shows the improvement when the maximum frequency is reduced to 0.45.

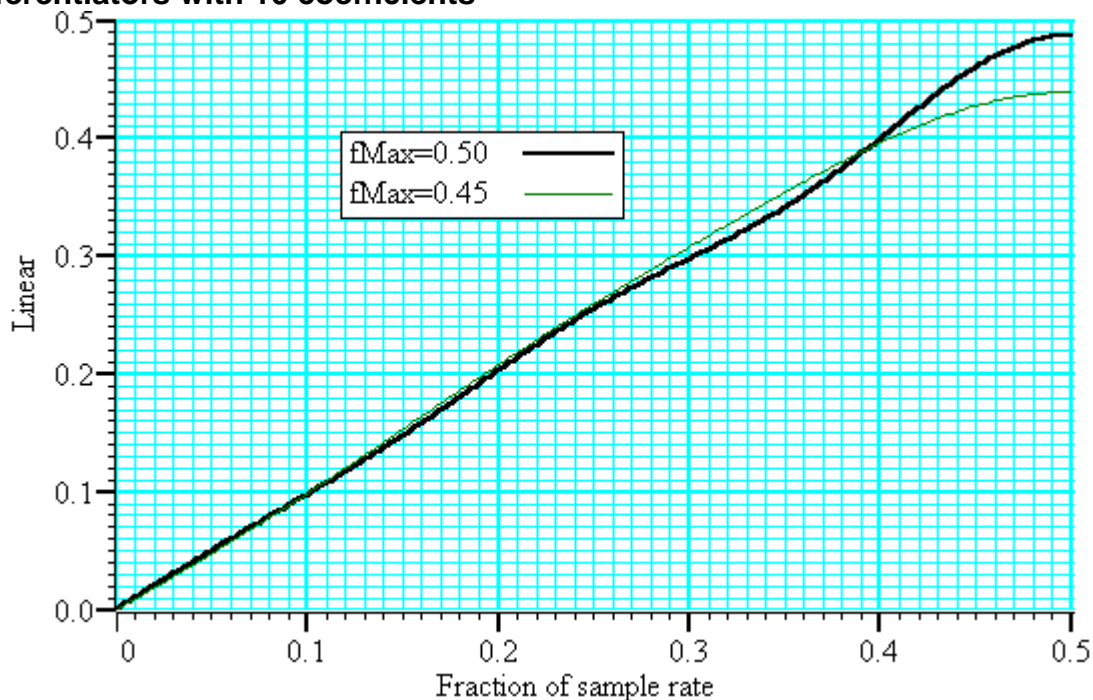
**Differentiators with 31 coefficients**



If you must span the full range, use an even number of coefficients. The graph below shows the improvement you get with an even number of coefficients. The ripple for the 0.45 case is about the same with 10 coefficients as for 31.



## Differentiators with 10 coefficients



```
var prm[4][1]; ' 1 bands for differentiator
var coef[10]; ' 10 coefficients needed
' band start band end slope weight
prm[0][0]:=0.00; prm[1][0]:=0.45; prm[2][0]:=1.0; prm[3][0]:=1.0;
FIRMake(2, prm[1][], coef[1]);
```

## Hilbert transformer

A Hilbert transformer phase shifts a band of frequencies from  $F_{\text{low}}$  to  $F_{\text{high}}$  by  $\pi/2$ . The target magnitude response in the band is to leave the magnitude unchanged.  $F_{\text{low}}$  must be greater than 0 and for the minimum magnitude overshoot in the undefined regions,  $F_{\text{high}}$  should be  $0.5 - F_{\text{low}}$ . The magnitude response at 0 is 0, and if an odd number of coefficients is set, then the response at 0.5 is also 0. This means that if you want  $F_{\text{high}}$  to be 0.5 (or near to it), you must use an even number of coefficients.

There is a special case of the transformer where there is an odd number of coefficients and  $F_{\text{high}} = 0.5 - F_{\text{low}}$ . In this case, every other coefficient is 0. This is of no help to Spike2, but users who write their own software can use this fact to halve the number of operations required to apply a filter.

It is extremely unlikely that a Hilbert transformer will be of any practical use in the context of Spike2, so we do not consider them further. You can find more information about this type of filter in *Theory and Application of Digital Signal Processing* by Rabiner and Gold.



# **19: Programmable signal conditioners**

# Programmable signal conditioners

The Spike2 signal conditioner control panel supports the CED 1902 Mk III and IV, The Power1401 gain option, the Axon Instruments CyberAmp and the Digitimer D360 signal conditioners. To use a programmable signal conditioner you must have selected support for it when you installed Spike2. You can re-install Spike2 to add a signal conditioner if you did not do this originally.

You can open the conditioner control panel from either the sampling configuration channel parameters dialog (when the channel type is waveform or WaveMark) or from the **Sample** menu. The 1902 and CyberAmp signal conditioners are controlled through a serial port which is set in the Edit menu Preferences... option.

## What a signal conditioner does

A signal conditioner takes an input signal and amplifies, shifts and/or filters it so that the data acquisition unit can sample it effectively. Many input signals from experimental equipment are too small, or are masked by high and or low frequency noise, or are not voltages and cannot be connected directly to the 1401.

Signal conditioners may also have specialist functions, for example converting transducer inputs into a useful signal, or providing mains notch filters. The CED 1902 has options for isolated inputs and specialised front ends include ECG with lead selection, magnetic stimulation artefact clamps and EMG rectification and filtering. The only option for the Power1401 is Gain. You should consult the documentation supplied with your signal conditioner to determine the full range of capabilities.

## Serial ports

The basic communication parameters are set in the Edit menu Preferences dialog. Most supported programmable signal conditioners are controlled through communication (serial) ports. No port is used if the conditioner support is not loaded or **None** is selected in the preferences or for the Power1401 gain option. Check the *Dump errors...* box to write diagnostic messages to CEDCOND.LOG in the current folder. Do not check the box unless you are having problems with a conditioner as it slows Spike2 down.

## Control panel

The control panel is in two halves. The left-hand half holds the controls that change the conditioner settings, the right-hand half displays data from the conditioner. The right-hand half is omitted if Spike2 is sampling data, or if the 1401 is not available for any other reason.

If the right-hand half is present, the Volts check box causes the data to be displayed in Volts at the conditioner input in place of user units as defined by the Channel parameters dialog. The number at the bottom right is the mean level of the signal in the area marked above the number.

Signal conditioners differ in their capabilities. Not all the controls listed below may appear for all conditioners. The controls are:

### Port

This is the physical 1401 port that the conditioner is attached to. If you open the conditioner dialog from the channel parameters dialog you cannot change the port.

### Input

If your signal conditioner has a choice of input options, you can select the input to use with this field. The choice of input may also affect the ranges of the other options.

### Gain

This field sets the gain for the signal selected by the Input field. Spike2 tracks changes of gain (and offset) and changes the channel gain and offset in the sampling configuration to preserve the y axis scale. You should

adjust the gain so that the maximum input signal does not exceed the limits of the data displayed on the right of the control panel. When Spike2 is sampling, the gain and offset are fixed once you have written data to disk.

This is the only editable field for the Power1401 ADC Gain option.

### **Offset**

Some signals are biased away from zero and must be offset back to zero before they can be amplified. If you are not interested in the mean level of your signal, only in the fluctuations, you may find it much simpler to AC couple (1902) or high-pass filter (CyberAmp) the signal and leave the offset at zero. If Spike2 is sampling you cannot change the offset once data has been written to disk.

### **Low pass filter**

A low-pass filter reduces high-frequency content in your signal. Filters are usually specified in terms of a corner frequency, which is the frequency at which they attenuate the power in the signal by a factor of two and a slope, which is how much they increase the attenuation for each doubling of frequency. Sampling theory tells us that you must sample a signal at a rate that is at least twice the highest frequency component present in the data. If you do not, the result may appear to contain signals at unexpected frequencies due to an effect called aliasing. As the highest frequency present will be above the corner frequency you should sample a channel at several times the filter corner frequency (probably between 3 and 10 times depending on the signal and the application).

You can choose a range of filter corner frequencies, or you can choose to have the data unfiltered (for use when the signal is already filtered due to the source).

### **High pass filter**

A high-pass filter reduces low-frequency components of the input signal. The high-pass filters area is specified in the same way as low-pass filters in terms of a corner frequency and a slope, except that the slope is the attenuation increase for each halving of frequency. If you set a high-pass filter, a change in the mean level of the signal will cause a temporary change in the output, but the output will return to zero again after a time that depends on the corner frequency of the filter. The lower the corner frequency, the longer it takes for mean level change to decay to zero.

### **Notch filter**

A notch filter is designed to remove a single frequency, usually set to the local mains power supply (50 Hz or 60 Hz, depending on country).

The remaining options are for the 1902 only:

### **AC couple**

This is present for the 1902 only, and can be thought of as a high-pass filter with a corner frequency of 0.16 Hz. However, it differs from the high-pass filters as it is applied to the signal at the input; the high-pass filters in the 1902 are applied at the output.

### **Trigger source**

The 1902 provides two conditioned trigger inputs, and one output. This control selects which of the inputs is connected to the output.

### **Filter type**

A 1902 mk IV with digital filters has extra fields that allow you to set the low-pass and high-pass filter types. You can choose Butterworth or Bessel characteristics and 2 pole or 3 pole filters.

### **Reset Calib**

This button is intended for use in the initial setup stage, before you start sampling. It adjusts the sampling configuration scale and offset so that the signal is displayed in units of Volts, millivolts or microvolts, as appropriate, depending on the gain that you have selected.

## Reload 1902

This only exists for the CED 1902. The 1902 is controlled by a serial interface and to set the complete state of it would take a noticeable time. To avoid this, we keep a record of the state that has been commanded and only transmit changes. If you click this button, we transmit all the commands to set every feature of the 1902 to match the control panel. This is provided so that you can recover a 1902 that has been powered down by accident.

## Setting the channel gain and offset

If you change the gain or offset in the control panel, Spike2 will adjust the channel gain and offset in the sampling configuration to compensate so as to keep the y axis showing the same units. This means that if you change the gain, the signals will still appear in the same units in the file. However, the first time you calibrate the channel you must tell the system how to scale the signal into y axis units.

### The easy way

If you want to set your channels to display in Volts, millivolts or microvolts, there is an automatic and easy way to do it:

1. Open the Sampling Configuration dialog.
2. Select the waveform or WaveMark channel and open the channel parameters dialog.
3. Click the **Conditioner** button to open the conditioner control panel for your signal conditioner.
4. Change the controls so that the largest signal you want to record will fit in the display window without clipping.
5. Click the **Reset Calib** button in the control panel, then **OK**.

You will find that the settings in the channel parameters dialog have been adjusted so that when you record, the channel will display in the correct units.

### The general case

If you must calibrate your input in other units than Volts, millivolts or microvolts, you have to do it manually. For example, to set up the y axis scales in microvolts by hand you could do the following:

1. Open the Sampling Configuration dialog.
2. Select the waveform or WaveMark channel and open the channel parameters dialog.
3. Set the Units field of the Channel parameters to  $\mu\text{V}$ .
4. Set the Input in Volts  $\times$  field to 1000000.
5. Click the **Conditioner** button to open the conditioner control panel.
6. Adjust the gain to give a reasonable signal.
7. Close the signal conditioner control panel.

You only need do steps 3 and 4 once. Any subsequent change to the conditioner gain (but not while sampling, see below) will adjust the channel gain to leave the units in microvolts.

For the more general case where you have a transducer that measures some physical quantity (Newtons, for example) and it has an output of 152.5 Newtons per mV. If you wanted the Y axis scaled in Newtons, you would replace steps 3 and 4 above with:

3. Set the Units field of the Channel parameters to N.
4. Set the Input in Volts  $\times$  field to 0.1525.

To work this out you must express the transducer calibration in terms of Units per Volt (in this case Newtons per Volt).

If you have set an offset in the conditioner, and you want to preserve the mean signal level, you should null it out by changing the offset in the Channel parameters dialog.

### Changing gain or offset while sampling

Ideally, you will set up the conditioner settings before you start to sample, and not change them during sampling. However, if you have set the initial gain too high or too low, or you need to null out an offset, you may have to change the settings during.

Spike2 waveform channels are stored as 16-bit integer values that are converted into real user units by a scale factor and an offset. The scale factor and offset are set for the channel; there is no concept of changing them during sampling. We allow you to use the conditioner control panel during sampling, but if you change the scale and/or offset, this does not automatically change the channel scale and offset as this would invalidate all previously recorded data.

If you have a TextMark channel enabled, each time you make a change, a new marker will be added to the channel with a marker code of 2 and the associated text will be something like:

```
ch 2 gain 100, offs 0.001
```

You could use this information offline to recalibrate the data.

If you do not have a TextMark channel enabled during sampling, the keyboard marker channel gets an event with code 2, so you know that a change was made.

## Conditioner connections

Spike2 can control multiple signal conditioners, but they must all be of the same type (selected when you install Spike2). Each conditioner has a unit number starting at 0 and each unit conditions the same number of data channels (1 for 1902, 8 or 2 for the CyberAmp, 8 for the Digitmer 360). For both the 1902 and the CyberAmp, if the conditioner unit number is  $u$  and there are  $n$  channels per conditioner, it conditions 1401 ADC channels  $u*n$  to  $u*(n+1)-1$ . It is up to you to make the correct connections.

Basic communication and connection information is stored in the registry in the key `HKEY_CURRENT_USER/Software/CED/CEDCond`. For each conditioner type that needs to store information there is an additional sub-key, currently 1902, CyberAmp, D360, Power1401. This sub-key holds integer variables:

- Port** If needed, the communication port number in the range 1 to 19. The default value is 1.
- Dump** Optional. Set to 1 to write the file `CECOND.LOG` holding diagnostic messages to the folder that Spike2 ran from.
- First** Optional, default 0. This sets the first 1902 or CyberAmp unit number to search for. Use `First` to save time waiting for missing units to time out.
- Last** Optional, default 0. Searches stop when there is no response from a unit number greater than `Last`. Use `Last` to skip over missing units.

The `Port` and `Dump` values can be set in the Edit menu Preferences option. Before revision 7.09 this information was saved in the file `CECOND.INI` in the system folder. If the registry does not contain the information, we read the `.INI` file. The file format is:

```
[General]
Port=COM1
Dump=1
First=1
Last=3
```

### CED 1902

The CED 1902s unit numbers are set by an internal switch pack; multiple units usually have the channel number as a label on the front panel. Multiple units must have different unit numbers. Each 1902 output should be connected to the 1401 ADC input with the same number as the 1902 unit.

Usually the 1902s unit numbers start at 0 and are contiguous, in which case you do not need to set `First` and `Last`. As an example of a more complicated situation, let us suppose you have unit numbers 4, 5, 6, 7, 12, 13, 14 and 15. In this case, set `First` to 4 and `Last` to 15 for ADC channels 4-7 and 12-15. The `Try1902` test program has options to set `First` and `Last`.

### CyberAmp

The CyberAmp has a device address (unit number) set by a rotary switch on the rear panel; multiple units must have different addresses. There are two types of CyberAmp: 8-channel and 2-channel. If you have multiple units, they must all have the same number of channels, or all the 8-channel units must have lower device addresses than all the 2-channel units and the channel mapping behaves as if all units had 8 channels.

Normally you will have one 8-channel CyberAmp, and you will give it address 0 to support ADC channels 0 to

7. In this case you do not need to set `First` and `Last`. If you want to connect it to channels 8-15, you would set both `First` and the device address to 1 and then you could connect it to channels 8-15.

Here is a more complex example with three CyberAmp devices, two with 8-channels and one with 2-channels. The table shows some possible configurations (assuming you have 32 ADC inputs):

| CEDCOND.INI |      | 8-channel unit 1 |       | 8-channel unit 2 |       | 2-channel unit 3 |       |
|-------------|------|------------------|-------|------------------|-------|------------------|-------|
| First       | Last | Switch           | ADC   | Switch           | ADC   | Switch           | ADC   |
| 0           | 2    | 0                | 0-7   | 1                | 8-15  | 2                | 16-17 |
| 1           | 3    | 1                | 8-15  | 2                | 16-23 | 3                | 24-25 |
| 2           | 3    | 2                | 16-23 | 3                | 24-31 | -                | -     |

### Digitimer D360

Install the Digitimer-supplied D360 support first and verify that it is working by running the Digitimer D360 Client application. The Digitimer D360 Client software must be at least version 4.7.0.0; upgrade to the latest version if it is older. The D360 connects to the computer with a serial line (COM port), however, our code has no control over the port; this is configured by the Digitimer code. The only registry option is `Dump`.

Multiple D360 amplifiers can be used; each has 8 channels numbered 1 to 8. Channels 1 to 8 of the first D360 are assumed to be connected to ADC inputs number 0 to 7, the second D360 drives ADC ports 8 to 15 and so on. The amplifier numbers are determined by the order in which they are daisy-chained together, they should be connected in serial-number order with the lowest-numbered amplifier connected directly to the PC, the next lowest-numbered amplifier connected to the first amplifier and so on.

### Power1401 with programmable gain

Spike2 will automatically detect the channels. The only registry option is `Dump`.



## **20: Test and utility programs**

# Test and utility programs

Spike2 includes programs that recover data from damaged 64-bit and 32-bit data files (often due to loss of power during recording) and a program to test that a 1401 is operating correctly.

## The S64Fix data recovery utility

The S64Fix program scans 64-bit (.smrx) data files to recover data. Use SonFix for 32-bit (.smr) files. S64Fix can repair two types of damaged files:

- 1) Files where sampling was interrupted and the file was not closed properly. In this case the file headers will hold sensible information, but it will likely relate to some earlier state of the sampling. Spike2 can detect this situation when you attempt to open such a file and will refuse to let you continue as any attempt to modify the file could overwrite valid data.
- 2) Files that have been damaged. If the damage is to the start of the file, the file may well refuse to open.
- 3) Files written before Spike2 version 8.02 where a channel has been deleted and reused. These files open without any problem, but data in the affected channel(s) can appear to be missing and may display differently, depending on where you start.

64-bit son files are designed to be relatively easy to recover. As long as the file header can be read, we can read through a file block by block and save data for blocks that appear undamaged and ignore blocks that do not make any sense to us. If a file header is damaged, as long as we can read the header from a similar, undamaged file, we can use that to recover the data from the damaged file.

## Recovering from interrupted sampling

Run S64Fix Open the program to display an empty window

Browse... Select the damaged file. The Progress section of the dialog will display:

Open file: MORE\_DATA - extra data on file end; maybe closed incorrectly

The son64 library can detect that the file is larger than the size stored in the file header. If there is a different message, for example WRONG\_FILE, skip to the second Browse... section, below.

New File You can now choose a name for a new file; do NOT use the same name as the file you are recovering. We scan the original file and write extracted data to the new file. If all goes well, the Progress bar will appear, showing progress as we read the source file. The Progress area will display any problems discovered and after reading through the entire file, a list of channels and the number of recovered blocks in each channel will appear. If all is well you can use the Restart button to recover a different file or Close to end the recovery session.

There can be an appreciable delay after fixing a file. This is because when the new file is closed, we flush all buffered data in the operating system to the disk file, and this can be many hundreds of megabytes of data (depending on the free memory in your system).

If the Progress section displays:

Open file: WRONG\_FILE - attempt to open wrong file type

this means that the file you selected was not recognised. If you know that it really is the correct type then the file header is damaged. In this case you should follow the instructions for Recovering a damaged file.

## Recovering a damaged file

Run S64Fix Open the S64Fix program to display an empty window

Browse... Select the damaged file. The Progress section of the dialog may display:

Open file: WRONG\_FILE - attempt to open wrong file type

If it does and the file really is a data file of the correct type, the file header has been damaged. In

this case click **Browse...** again.

- Browse...** If the damaged file does not open you can read the file header information from another file that holds the same set of channels. Click the **Browse...** button and open a suitable file; it must have the same channel numbers and channel types as the file you wish to rescue. If the file opens without error, you will see the file in the **Replacement header file** section of the dialog.
- New File** Click here to set the name of the recovered file and then process the damaged file and write recovered blocks to the new file.

## Recovering reuse of a deleted channel before Spike2 8.02

Run S64Fix Open the S64Fix program to display an empty window

**Browse...** Select the damaged file. The file will appear to open normally.

**New File** Click here to set the name of the recovered file and then process the damaged file and write recovered blocks to the new file.

The problem with these files is that the internal tree that allows us to locate channel data efficiently does not have the correct start times of each data block. This causes the lookup process that locates the data to fail. The usual result is that the file appears to have missing data. The repair process ignores the internal tree, and links together the data blocks that it finds in the file. It is possible to get data returned beyond the end of the actual recorded data; if this should happen you can ignore it, or fix the problem by exporting the result to a new file, omitting the unwanted data at the end of the channel.

## How fixing works

A 64-bit son data file is composed of data blocks of a fixed size. There are large blocks (typically 64 kB in size) that hold either header information or data for one channel. There are also smaller blocks (typically 8 kB in size) that hold channel indexing information. The block sizes are chosen to balance the requirements of high performance and reasonable memory usage. The block sizes are coded in the file header; it is possible to build the software for different block sizes for different applications, though we have no plans to do this at the moment.

The small blocks are grouped together to use a space the same size as a large block. This means we can scan a data file sequentially by reading through it in steps equal to the large block size. Each block starts with information that identifies the block type and the data channel to which it belongs (header blocks have a channel number of 65535). As the index blocks are used to locate data very quickly when reading, we can ignore them when scanning for data. This means that if the file index system gets damaged, which would stop Spike2 reading through a channel, we can still recover any undamaged data from the file.

For the scanning process to work correctly, we need a file header that as a minimum, identifies the channel numbers and channel types in the file. Given this information, we read through the file, block by block. For each block that has a recognisable header we check that the data in the block is plausible, and if this is the case, we write the data to the output file.

At the time of writing we have very little experience of damaged data files, other than files that we have deliberately damaged by switching off the system power while sampling (do not try this yourselves as you run the risk of trashing your system). If other systematic sources of error show up that require different fixing strategies we will apply them in the future.

## What we cannot recover

There are several types of data that are gone forever...

### **Data that was never written to the disk**

When we sample data, there is a data buffer of typically 8 MB that holds the most recently sampled data on

each channel. This allows you to set up triggered sampling scenarios where data is usually not saved to disk, but which is marked for saving based on some criterion, often an external trigger or real-time data analysis. The buffering used for this is "circular", that is new data overwrites the oldest buffered data. We also have as list of times associated with each buffer for which data is to be saved.

When data is ejected from the circular buffer and is marked as wanted it is accumulated in a channel "write buffer". Each time a write buffer becomes full, it is written to disk (or more accurately, written to the operating system filing system). This means that during sampling, each channel has quite a bit of buffered data in memory. There is a write buffer of up to one whole block of data, holding data that is wanted and there is also a circular buffer holding data that may or may not be wanted.

The problem with this is that should the system power fail or your computer crash, you will lose all data in the circular buffers and the write buffers. To mitigate this problem, you can chose to commit data to disk at regular intervals. Each time a commit happens, all data in the circular buffers that is marked as wanted is sent to the write buffer (which may cause it to be written if it fills), then anything remaining in the write buffer is also written (creating a partially full buffer on disk). Normally we only write to disk when we have full disk blocks to write; setting commits every few seconds will be inefficient, especially in cases where you have a large number of data channels (with n channels you force a minimum of n extra writes).

### **Data that has been overwritten with other data on disk**

This is most unusual on modern systems, but it was quite common in the early days of Windows when rogue programs (or even the operating system) could trash the filing system. If you do suffer from a disk crash and need to recover the disk, STOP USING IT. Any operation that writes data to the disk will reduce your chance of recovering disk blocks. A more common problem is to delete a file you wanted. You can often get such files back from the trash can. If you have done a thorough job of deleting (by asking the system to not save the deleted file), an expert may still be able to recover the files AS LONG AS YOU STOP WRITING DATA TO THE DISK. Such recovered files may have a few missing or overwritten blocks, but S64Fix should still be able to get data out of it.

### **Corrupted media**

Writing data to any device has an error rate... (though it will likely be vanishingly small). Put another way, with any device, if you write enough data you will get errors. Usually, errors are hidden by automatic error recovery systems; some drives have a S.M.A.R.T. feature that claims to give predictive reliability information. The general rule seems to be that if you ever see bad sectors on a drive, you are likely to see more. A new drive (even allowing for the time to move everything) is far cheaper than the time and effort to recover data from a bad device.

There are many programs available (some with free trial periods) that will attempt to get back data from hard drives and optical media. Search the interweb for "Recover data from" followed by the type of your storage medium for a long list of products.

As long as the recovery program does not try to "help" you by cutting out bad blocks, once you have a file that is the same as the original but with a few blocks of rubbish in it, S64Fix should be able to read back the undamaged data.

A popular way to lose data is to write it to optical media and then leave them in direct sunlight... If you archive to optical media, store them somewhere dark and cool.

## The SonFix data recovery utility

The SonFix utility program recovers damaged 32-bit (.smr) Spike2 data files. See the S64Fix program to recover 64-bit son files. A Spike2 data file has a header followed by a channel description list, followed by data blocks. Each data block has links to the previous and next data block of the same channel. If a file becomes damaged, these links can be broken, rendering the file useless. The SonFix program scans the file, rebuilds the links for each channel and corrects the file information in the file header. There are three common types of damage that occur to Spike2 data files:

1. The file is damaged as it is sampled. This can happen if your system loses power. The next time you run Spike2 it will tell you where the data can be found and recommend that you run SonFix to repair the file.
2. The file is damaged due to some other disk accident (such as a disk crash or when archiving and restoring files to tape, CD, DVD or some other removable media). As long as the damage is not at the start of the file, the file can usually be fixed and undamaged data recovered.
3. You can generate event or digital marker data with multiple data points sharing the same data file time (when the actual time difference between items is less than one file time unit). The SON file system is not able to completely deal with this situation which may cause problems during analysis of your data files. Spike2 6.10 and later versions will warn you if this situation exists when sampling stops; SonFix can usually fix the problem by slightly adjusting the times of overlapping items.

SonFix relies on the file header and channel list being intact. However, if the program does not recognise the file as a Spike2 data file, don't despair. It is possible to patch the file header with a header from a similar file, and then recover the data. Contact the CED Help desk if you are in this situation.

If you record a very large file with the Big file option (many GB in size), it is possible that it may be too large for SonFix to recover it. This happens when the memory required to hold a list of all the data blocks (and potential data blocks) in the file exceeds the addressable memory available to a 32-bit program. We have no work around for this condition.

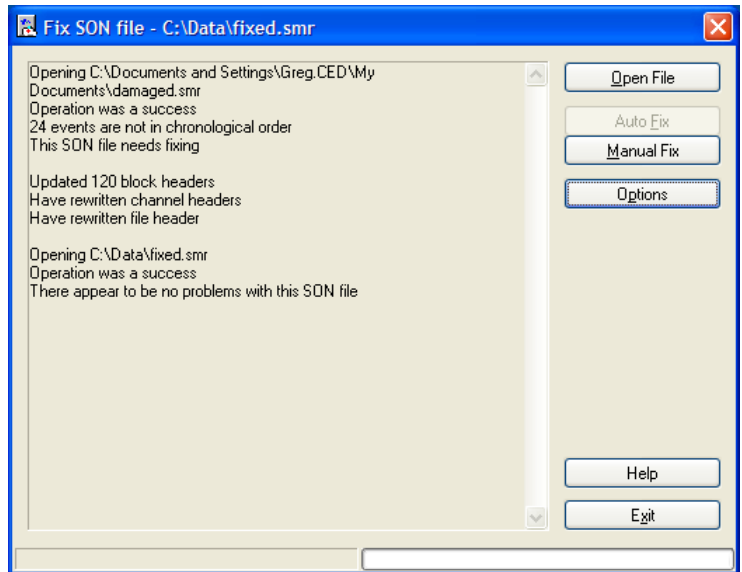
## Using SonFix

If you have a disaster, check that your disk drive is not damaged. Writing more data to a damaged drive is likely to cause further problems and may render your data unrecoverable. If the hard disk is damaged you must repair it with a suitable disk utility program or find someone who can do this for you before running SonFix.

Next, find the data file. If you have to use a disk repair utility, you may find it useful to know that the first two bytes of a data file hold the file revision followed by the ASCII characters "(C) CED 87".

In most cases there will be no disk damage and the file will be present with a non-zero size. If your data is very valuable, you might want to save a copy of the damaged file in case you need to send us a copy for a difficult recovery. Run SonFix by double clicking its icon (it is in the same folder as the Spike2 program). Click the Open File button and select the file to repair and SonFix will tell you if it can repair the file. Click the Auto Fix button to carry out the repair. If SonFix reports that the file was repaired without any problems, it should be possible to open it with Spike2. As a final cleanup, you can use Spike2 to export all the data in the repaired file to a new file; this will eliminate any unused file space.

The Options button opens a dialog that controls the tests carried out on a data file to detect damage and sets how event and marker channels are checked and repaired. If the data file is heavily damaged, it may be



necessary to use the Manual Fix option. Contact CED if you have problems fixing a file or using the manual fix tools.

### What cannot be recovered

Data that was not written to the physical disk cannot be recovered. To keep the system responsive and to make high speed sampling efficient, Spike2 buffers data in memory as much as it can. The operating system also buffers data in memory before writing it to disk. However, with data in memory, a power loss or system crash will lose data.

If you use the Flush data to disk option in the sampling configuration, you can protect yourself against disaster at the cost of some loss of performance. If you do not use the flush data option you increase the risk of data loss if your system goes down.

## Batch Processing

There are two ways to test or fix more than one file at a time:

1. Use the Open File button on the main window to show a standard File Open dialog box in which you can select as many SON files as you wish to test or fix.
2. Drag files from Windows Explorer or My Computer and drop them on the SonFix window is the other way of testing or fixing several files at the same time. You can also drag a directory onto the SonFix window, and it will test or fix all the SON files (files with a .smr extension) in that directory, and optionally all the SON files in any sub-directories (this setting can be changed from the Options window).

If you want files to be fixed automatically, you can specify this in the Options window. Fixed files have Fixed added on to the start of the file name. The fixed files will, by default, be placed in the same directory as the damaged files, but this can be changed from the Options window.

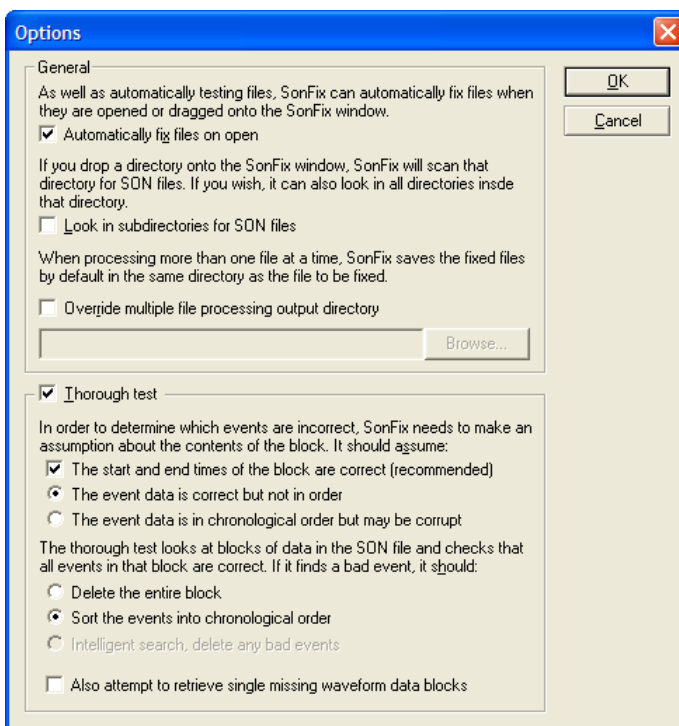
After SonFix finishes testing and optionally fixing your data files, it displays a list of the files that needed fixing, and a summary of the number of files tested and fixed.

## SonFix Options

To customise SonFix, select Options from the main SonFix window. As well as the options described above, there is also a section in the options dialog box entitled Thorough Test, which enables a complete test of the times of event and marker data. If selected, the thorough test is performed on files after the main test, and checks all events in the data file to make sure that they are in the correct order. To help with the error detection, you can select various assumptions that SonFix should make when it attempts to recover data from a damaged file:

### Block start and end times are correct

You should normally select this option. It tells SonFix to assume that only the actual event data is corrupt, not the data block headers. If, when you use this option, you find that a great deal of event or marker data is deleted, it would be worthwhile trying to fix the file without this option to see if that helps.



### **Event data is correct but not in order**

#### **Event data is in chronological order but may be corrupted**

You can choose between one of these two possibilities, which determine both how the event data is checked and also how it may be fixed. There is no correct choice here; you may simply have to try both options to see which gives the better fix. To start with, selecting data in order but corrupted is probably a better guess.

If SonFix detects event or marker data corruption, it can fix problems in these ways:

- by deleting the block containing the corrupt events (a good idea for heavily damaged files where you care more about the waveform data than event or marker data), or if the other repair option fail.
- by sorting the events into numerical order or, in the case where two events have the same time, adjusting the time of the second event to be just later (for example for a file containing RealMark, WaveMark or TextMark channels where you care more about the data they contain than the times they are stored at). If adjusting a time causes an event to be at the same time as the next event, then the next event is also moved. This repair mechanism is only available if you are assuming that event data is out-of-order but not corrupt.
- by doing an intelligent search for bad events and deleting the minimum number of events to leave the data in order. This is usually the best option. It is only available if you are assuming that event data is in order but corrupted.

#### **Also attempt to retrieve missing single waveform data blocks**

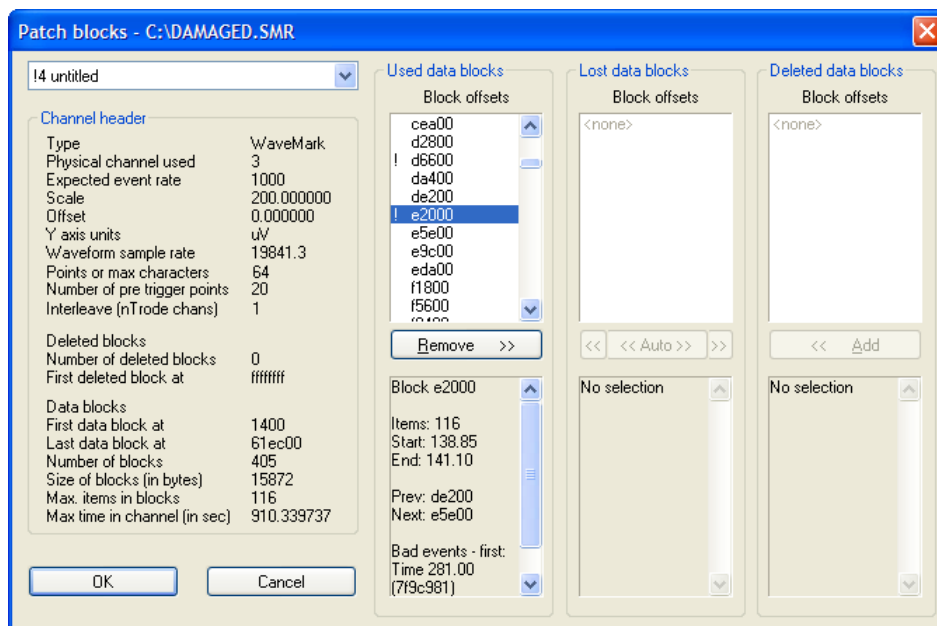
If a waveform data block has a corrupted header, it may not appear in any block list. If the used list has a gap where the forward and backward pointers both agree on the position of a missing block, and no other channel claims it, SonFix can recreate the block header. This is possible because waveform data is sampled at a known rate; this cannot be done for event or marker data. This option takes advantage of this to retrieve single missing blocks of waveform data. Retrieved data blocks may contain bad waveform data, but recovery removes gaps in the channel, which could be useful.

#### **Options to select when fixing duplicated event times**

If you are here because Spike2 reported duplicated event times after sampling, the options to select are: *Block start and end times are correct*, *Event data is correct but not in order* and *Sort the events into chronological order*.

## **Manual Fix**

The Manual Fix button leads to the Patch blocks dialog in which you can inspect the lists of used, deleted and lost data blocks for each channel and other channel details. You can also move blocks between these three lists for each channel. Unless you are certain that you know what you are doing it is best to only use Auto Fix; the manual fix tools are intended for use primarily by CED personnel or under CED direction.



The Patch blocks dialog shows information for one channel at a time; the channel selector is at the top left of the dialog. Channels with detected problems have an exclamation mark before the channel number. You should be able to ignore channels without such an indicator.

The channel header information is displayed on the left side of the dialog. The information shown includes general channel information such as the channel type and sampling rate, information about any deleted blocks (these are disk blocks that held data when the channel was deleted at some point in the past, and have not yet been reused) and details of the channel data blocks themselves.

The detailed data block information is shown at the right of the dialog. All data blocks are multiples of 512 bytes in size. When SonFix scans the data file, it searches for data that looks like the start of a block at each 512 byte boundary in the file. The block headers hold the channel number to which the block belongs. When it finds a block, it can use the block size information in the file header to predict where the next block could start. The scan of the file allows SonFix to build up block lists for each channel:

### Used data blocks

Used data blocks are blocks of data that are correctly linked into the channel and that do not show any problems. For a channel that is OK, all of the data blocks will be shown in the Used list (with possibly some in the deleted list).

### Lost data blocks

Lost data blocks are blocks that apparently make up part of the channel data (because they have this channel number in the block header) but which are not connected properly to the rest of the channel data or the header information does not make sense. Fixing a channel usually comes down to moving these lost blocks into either the Used list or into the Deleted list.

### Deleted data blocks

Deleted blocks are blocks forming part of the deleted channel data. When you delete a channel in a data file, the chain of blocks belonging to the channel are placed in the channels deleted block list. If you reuse the channel with the same block size, the deleted blocks are reused. If you re-use the channel with a different block size, the deleted blocks are abandoned. The presence of deleted blocks does not indicate a problem.

Each set of information shows a list of data blocks above, and information about a single (selected) data block below. The list of data blocks shows each data block's address within the disk file as a hexadecimal (base sixteen) number. Blocks that are deemed invalid (either in relationship to the adjacent blocks in the list or as a result of internal consistency checks) are shown with a preceding exclamation mark, blocks that make up part of a coherent chain of blocks descending from an initial block are shown indented to the right with the initial block not being indented. The block information shown in the area below shows the number of items, the time range covered by the block, the previous and next blocks in the chain and details of any errors detected.

Use the buttons at the bottom of the block lists to move blocks between lists. The Remove button below the



Used list moves the selected block into the deleted list, while the Add button in the deleted list moves a block into the Used list. However these options are rarely required compared to the buttons below the Lost list. The button marked '< <' moves a block, or set of blocks, into the Used list. Normally it will only move the selected block but, if this block is the start of a coherent set of blocks all shown indented to the right, then all of the blocks within this coherent set are also moved. The button marked '> >' does the same thing, but it moves blocks into the Deleted list.

The third button is marked '< < Auto > >'. This does the following:

1. Attempts to move all of the lost blocks into the used list, as long as this will not result in a corrupt or invalid set of blocks.
2. If step 1 fails, finds the largest single coherent set of blocks in the lost list that can be moved into the used list without problems and moves it.
3. If no blocks can be moved into the used list without problems, all of the lost blocks are moved into the deleted list.

This corresponds roughly to the process carried out by the Auto Fix button in the main SonFix window, though the Auto Fix button repeatedly executes the three steps shown above until the lost list is empty.

It is not possible to describe how to use the Patch blocks dialog completely as the steps required will vary according to the file damage. Roughly speaking, you should attempt to first find out which lost blocks are hopelessly corrupted and move them to the deleted list, then move all of the others to the used list. Once all of the rubbish is in the deleted list the Auto button should do this for you. It is very strongly recommended that you make a backup copy of the damaged data file before attempting a manual fix so that mistakes can be corrected.

## Try1401 test program

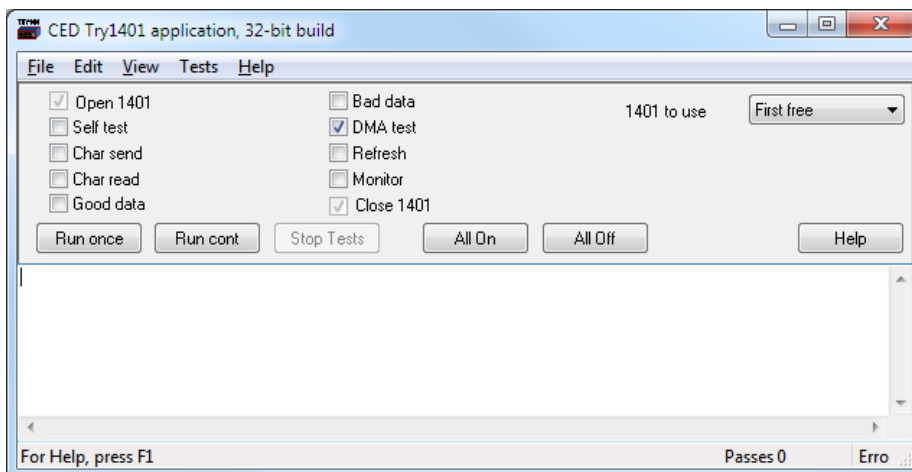
If you are having problems sampling data, and you suspect that the problem lies with the 1401, interface card or device driver, then it is well worth running the Try1401 program, found in the same folder as Spike2 (the program file is called `Try1432.exe` on Windows systems).

The Try1401 program exercises the 1401 in much the same way as Spike2 does, but it also checks every byte of data transferred and reports errors in a way that is useful to CED engineers, particularly when there are data transfer problems. Unfortunately, the sternest test of data transfer we know of is the Spike2 program, so it is possible for Try1401 to pass a system that fails in Spike2 (but this is most unusual).

The Try1401 program can also run the 1401 self-test and decode any errors. These can vary from simple problems with the calibration of the 1401 inputs and outputs (usually not a serious problem), to serious internal errors associated with damaged components. You should select the self-test option if the 1401 flashes the Test LED on power up when no other external equipment is connected to the 1401 inputs.

## Running Try1401

Double click the program icon to start the program. You have a choice of test options and of running once or continuously. Errors are written to the text window, and can be copied, cut and pasted. If a single pass of the test doesn't show a problem, run the test continuously and leave it for several hours.



Most of the tests are to check the data path between the host computer and the 1401. If you have a problem with your 1401, the CED engineer who helps you to sort it out will most likely ask you to run this program and email the results. For a confidence check, the only option that is needed is the DMA test. The remaining tests help CED engineers to narrow down problems. The individual options are:

### Self test

This runs the internal 1401 self test and interprets the results. Please remove all connections from the 1401 except the power cord and the data interface cable as other connections can cause self-test failures.

### Char send

This checks the general data path from the host computer to the 1401. If you have a USB connection, problems indicate some sort of installation failure. For the other interface cards data corruption problems are rare, and indicate either a damaged data cable, or bent pins on the 1401 or the host interface card. A timeout error in this test usually indicates an interrupt problem.

### Char read

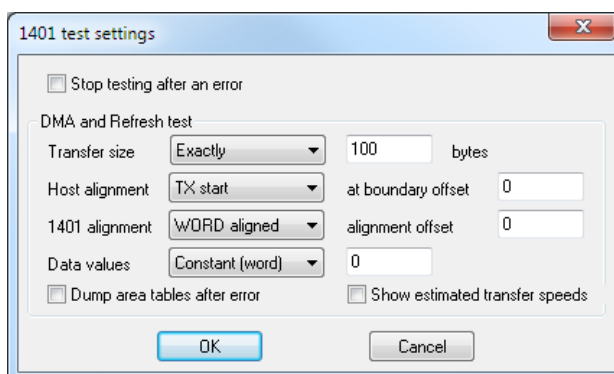
This is similar to *Char send*. It checks the general data path with an emphasis on reading back data.

### Good data/Bad data

These are further variations on the *Char send* and *Char read* tests. You can usually skip these tests unless a CED engineer asks you to run them.

### DMA test

This test checks out high-speed block data transfer between the 1401 and your computer. There is no point running this test if the *Char send* or *Char read* tests have failed. This test transfers data blocks of various sizes and alignments and checks that the data transferred correctly. Unfortunately, Spike2 is the sternest test of data transfers that we know of, but this test will usually detect any transfer problems. There are additional options in the Tests menu DMA settings... command:



You will not normally need to set any of the values in this dialog unless a CED engineer needs additional data

to diagnose a data transfer problem. The fields in the dialog are:

Transfer size can be set to: **Unconstrained**, **Exactly**, **Greater than** or **Less than**. For all sizes except **Unconstrained** you must provide a byte count. For all except **Exactly** mode, Spike2 chooses random sizes within the constraint you have imposed.

The data for transfer is held in memory. This memory lies in a continuous block of virtual address space. However, the physical memory that makes up this address space may be mapped anywhere in computer memory. This usually means that a data transfer is broken up into sub-blocks of contiguous physical memory. **Alignment** relates to the position of data blocks relative to starts and ends of these sub-blocks. This field only applies to ISA and PCI interface cards using DMA transfers; we have no knowledge of alignment for the USB interface. You can choose from **Unconstrained**, **TX start**, **TX end**, **RX start** and **RX end**. TX = transmit to 1401, RX = receive from 1401. The **at boundary offset** field sets the relative position of the transfer start or end to sub-block boundaries. You can set from -4095 to +4095 (but useful values are usually in the range  $\pm 100$ ).

Data values can be set to be **Random**, or you can choose to set values based on bytes, words (2 bytes) or longs (4 bytes) and the values can be a constant, an upward ramp or a downward ramp.

Check the **Dump area tables after error** box to print a lot of extra info about any failing DMA transfer. This can help a CED engineer to diagnose a fault.

**Show estimated transfer speeds** prints the approximate number of kB per second for each transfer. The rate depends quite strongly on the size of the transfer as there is a time overhead to setting up the transfers and detecting that they are done. There will always be some transfers of 250,000 bytes and these give the best estimate of the available maximum speed.

## File menu

There are options in the File menu to update the Power1401 and Micro1401 mk II or -3 flash memory. These are described in the Owner's manual for your 1401. You can get the downloads from our web site. See the About Spike2 page for links to the monitor downloads.

There is also the **1401 info...** command. This prints out information about the 1401 device driver and the 1401, for example:

```
1401 type = Micro1401
Monitor revision is 20.14
1 megabyte base memory
Micro1401 main card is 2501-01 C-211
Block transfers use DMA, multiple transfer areas
ADC channel sequencer is FIFO at 4 MHz
Supports up to 256 channels, 3uS ADC block
No extra ROM in spare slot
```



# **21: Multimedia recording**

# Multimedia recording

## The s2video application and file names

You can use the `s2video` application to record multimedia files automatically whenever you sample a Spike2 data file. You can run up to 4 instances of this application at a time. Each instance will record one video and/or one audio track. The quality of the recording (video resolution and frame rate and audio sample rate) will depend on the hardware and software that you have installed. You can use codecs to compress your multimedia data either on-line during capture or off-line after capture ends.

The multimedia data files are saved alongside the Spike2 data files. You can view these files within Spike2 using the View menu Multimedia Files command.

### File names

Spike2 samples data to a temporary file. When sampling ends and the user chooses to save the file, it is given a permanent name. The multimedia recorder tracks the Spike2 activity and names the data files to match Spike2. For example, if the final Spike2 name is of the form:

```
C:\Spike7\data\data23.smrx (or data23.smr)
```

and if there were two instances of `s2video` running, they would name their files:

```
C:\Spike7\data\data23-1.avi
C:\Spike7\data\data23-2.avi
```

### Availability of files

Multimedia files generated by `s2video` are not available for use until the Spike2 data file has been saved. If they require off-line compression, this is done as a background task so that data capture can continue. In this case, the multimedia files are not available for review until the compression task has been completed. Alternatively, you can store files uncompressed (they will likely be huge and take a long time to move between drives) and then use `avicompress` to compress them at a more convenient time.

### Timing and AVI files

An AVI data file is a container for streams of data; in our case we are interested in audio and video data.

Video data is stored as frames, either as key frames that contain all the data needed to render them, or as frames that hold the difference from the previous frame in some way. In a standard AVI file, the first frame starts at time 0, and the frames occur at a fixed time interval. There is an index in the file that locates the position and holds the size of each frame. If frames were dropped during sampling, they still get index entries (to maintain the file timing), but the size of the data is usually 0. Video source devices announce the frame rates that they support, and in theory, they deliver frames at this rate. In practice, they deliver frames at a rate that is close to this rate, but can differ from it by several percent. When we also take into account the fact that the timebase used by Spike2 is based on a clock in the 1401 device, there is clearly a problem.

To help to correct time drifts, we adjust the clock used to time the multimedia to agree with the Spike2 clock during sampling. However, this doesn't do anything to correct the frame rate stored in the data file. If timing of video frames is a problem, there are options in the Configuration menu to recalculate the frame rate based on number of frames and elapsed time. You can also set an initial time offset to account for fixed time offsets between the multimedia stream and the Spike2 stream.

If there is insufficient bandwidth in your system to write the multimedia stream to disk, data will be lost. In some cases, the system will be able to drop frames of data and still maintain the timing; in other cases this will make it impossible to line up the resulting AVI file with Spike2 data.

## System requirements

To run Spike2 data capture and multimedia capture simultaneously, you need a suitably powerful computer. In October 2011, a new system will likely have two or more processor cores, 2 or more GB of memory and a 500 GB hard disk and run Windows 7. This is the type of system you need. The software will take advantage of multiple processors and hyper-threading. Spike2 also runs on Windows XP and Vista.

The Multimedia data capture and replay software (used for video recording and playback) is based on Microsoft DirectShow, which is/was part of DirectX. A version of DirectX is probably already installed on your computer. The `dxdiag` utility will report the DirectX version and test it (open the **Start** menu, select **Run**, type in `dxdiag` and click **OK**). If you are running Windows XP you should check you have the latest version. You can download this from the Microsoft web site: go to the Downloads section and search for DirectX. The latest version in March 2008 was DirectX 9.0c. If you are running Vista or later you should be up to date with the latest versions. There is a replacement for DirectShow that Microsoft is shipping with Vista and later systems, but this is not supported in Windows XP, so we will not be using it until support for XP is no longer required.

To record video, you will need a web cam or a video camera. There is a very wide variety of these available. You will need to install drivers for your camera; these will normally be supplied with it. Alternatively, your system may recognise the camera when you plug it in and locate the correct drivers for you.

To record audio you need a sound card in your computer or a video device that also records sound.

If both your camera/audio device and 1401 link to the computer by USB, and your computer does not have independent USB ports, the two devices will compete for bandwidth. This can degrade the maximum sampling rate of the 1401 and the maximum frame rate of the video.

If your computer has both USB 1 and USB 2 ports, beware of plugging a video device (or a 1401!) into a USB 1 port. This severely reduces the bandwidth. Even modest video needs a bandwidth of many MB per second and USB 1 will limit you to just over 1 MB per second. For example, my web cam set to 320x240 at 30 frames per second generates 6.4 MB per second. If I plug it into a USB 1 port, it drops more than 80% of the data frames and the result cannot be synchronised.

## Problems

One of the good things about using DirectShow is that it allows the use of an extremely wide range of video and audio devices. Sadly, this also leads to problems. Although most companies ship good-quality software drivers for their multimedia products, not all ship DirectShow drivers. Ironically, almost all high-volume, consumer (i.e. cheap) web-cams and video cameras work, but some low-volume, specialised (i.e. expensive) devices do not. We regret that there is nothing we can do about this.

## Getting started

When you run the `s2video` application for the first time, there will be no devices selected. The window title **Spike2 Video Capture 1** means that this is the first instance of the application. If you open another, it will be number 2, and so on. The application stores information in the system registry for each instance. This includes the window position, the video and audio recording devices and any codecs that you select to compress the multimedia data so that it occupies less disk space.

Your first task is to choose the devices to record from. Click the **Settings** menu and then select the **Video Device** item. A new menu will pop-up with a list of the possible video sources plus **No Video Device**, which will have a tick next to it. If a device that you want to use does not appear in the list, make sure that it is switched on and connected. You do not have to select a video device; recording audio only is allowed. Once you select a video device, you will see a preview of the image in the video display area of the window. You can enable and disable the preview with the **View** menu **Preview** command or with the first button in the toolbar. If you select **No Video Device**, the window will reduce in size to hide the video area.

The next task is to select an audio device. Click the **Audio Device** item in the **Settings** menu to display a list of possible audio devices, plus **No Audio Device**. If you selected a video device that generates audio as an integral part of the video stream, the **From Video Device** item is enabled and can be selected. When you select an audio device, an audio level meter appears in the dialog. You do not have to select an audio device; recording video only is allowed.

You must set at least one device to sample data. Once you have set your devices, new items for Video Device Properties, Video Capture and Audio Device Properties are added to the menu.

This software is written to work with a very wide range of hardware devices. We cannot predict what you will see on the screen in the property dialogs for the devices as the device manufacturers define them. We can tell you the types of control to expect from our experiences working with a range of different cameras and sound devices. If you have problems setting up your video or audio device please refer to the documentation that was supplied with them. We will be able to offer general guidance only.

## Video Device Properties

This menu item appears in the **Settings** menu when you have selected a video input. What you see in this menu depends on the selected camera. It will probably include controls for image brightness, saturation, colour balance and saturation. It may also have controls for the frame rate (this will set the maximum frame rate that you can record at), monochrome or colour output and various settings to compensate for background illumination and to cope with fluorescent lighting.

If you can control the frame rate here, set the lowest rate that provides enough detail for your sampling tasks. If you do not need colour and you can request monochrome here, you will save disk space (in compressed images) if you select monochrome.

If your video device includes audio support, there may also be audio setup controls here. These may replace or extend the controls in the **Audio Device Properties** dialog.

You will not normally find controls for the image size or the image compression format here; these are usually in the Video Capture page.

If the Properties dialog shows disabled items that you would expect to be able to change, you may be able to enable these by checking the *Disable video during Properties dialog* button in the *Configuration* dialog.

## Audio Device Properties

This menu item appears in the **Settings** menu when you set an audio device. It opens a **Properties** dialog for your selected device. If the audio device has multiple inputs you can use this dialog to select the one to record. You can also set the volume level of the input. There are fields for stereo panning, treble and bass, loudness (bass boost) and to force monophonic sound from a stereo source.

With most common sound devices the majority of fields are grey and cannot be used. If the **Enable** field is not grey, it is important that you use the **Pin Line** field to display the input to record from and then check the **Enable** field to select that input. You should also make sure that the volume control (the vertical slider under the **Pin Line Input Mix** label) is not set to the bottom position, which is zero volume.

If the Properties dialog shows disabled items that you would expect to be able to change, you may be able to enable these by checking the *Disable video during Properties dialog* button in the *Configuration* dialog.

## Video Capture

This menu item appears when you set a video device. It opens a dialog in which you can set the frame rate. It may also allow you to control the image size and format. This may duplicate controls in the **Video Device Property** page.

The **Frame Rate** sets the frames per second produced by the camera. You will get fewer frames per second than this if the system cannot keep up or if you limit rate with the **Set Slow Frame Rate** dialog or with the `Spike2 MMRate()` script command. The lower the frame rate, the smaller the data file. Use the lowest frame rate that gives you enough information for your task. A higher frame rate uses up processor time and disk space. It is likely that a rate of 10 or even 5 frames per second is all you need for many applications. The range of allowed frame rates depends on the camera. Some cameras have a single, fixed frame rate.

The **Output Size** is the image resolution expressed as **horizontal x vertical** pixels. Please use the smallest image size that is suitable for your needs. The size of the output file is proportional to the product of the horizontal and vertical sizes. Doubling the resolution creates a file that is four times the size.



The remaining settings depend on the device. Unless you have a good reason to change them we suggest you accept the initial values.

### **Image format**

The image format specifies how the camera arranges and compresses the video stream from the camera. There are many different formats possible, all with differing benefits. It is possible that s2video may not work with all the formats your camera can produce. This is because the DirectShow system has to connect the camera source to the (optional) video compressor you have selected, and then to the AVI data file. To do this, it has to find a set of DirectShow filters that can do any format translation required. If a filter for a particular format is missing or the DirectShow software cannot figure out how to make the connection, the format cannot be used and you will get an error if you select it.

### **Troubleshooting video**

If you cannot get a video camera that has a DirectShow driver to work with s2video there may not be a lot we can do about it. If you are prepared to send us your camera and the driver software we may be able to track down the problem, but there is no guarantee that we can fix it. Please remember that a very large number of devices do work; we have to deal with a generic interface to a huge range of possible video devices and what is possible depends largely on the quality of the DirectShow device driver and any auxiliary DirectShow filters provided with the camera.

## **Set Slow Frame Rate**

This command opens a dialog in which you can set the maximum frame rate to capture and a slower rate to use when you do not need the rate set in the **Video Capture** dialog. The slower rate is also set by the `Spike2MMRate()` script command. We achieve these rates by throwing away frames captured by the camera.

If the source video from the camera comes is compressed, we cannot start the stream after a gap until a "key-frame" comes along (that is a frame that contains enough information to generate the entire image). Some devices generate separate outputs for Preview (uncompressed) and Capture (compressed); it can happen that the preview display shows exactly the frame rate you request (1 frame per second, for example), but the frames written to the file may be at different intervals.

DO NOT set the slow frame rate to the actual rate that the camera is producing. For example, if the camera generates 30 FPS and you set a slow rate of 30 FPS, you may reject every other frame due to slight timing irregularities. You will normally set the maximum frame rate to a rate much higher than the camera can generate.

## **Use Slow Frame Rate**

Use this command to swap between the slow frame rate and the maximum frame rate set in the **Set Slow Frame Rate** dialog.

## **Configuration**

The configuration command opens a menu in which you can choose a data compression strategy and also how you will adjust the data frame rate so the video stream stays synchronised to the times recorded in Spike2. It also has a check box for working around a problem we discovered in some older audio drivers that causes a system crash.

### **Video and Audio compression settings**

Compressing your data can reduce file size considerably; reducing a file to 2% of its original size is entirely possible, especially if the video data is relatively constant. Given that a raw video stream may be producing several MB a second, this is well worth having. The down side is that compressing data takes significant processing power. Ideally, you would like to compress the data in real time, but in practice this may not be possible. Compression is achieved by choosing a suitable codec. For both video and audio you can choose from:

### *On-the-fly compression*

For video, unless you have a powerful computer or specialised hardware, this may be too slow to be useful unless you are running at low frame rates or at relatively low resolution. If your computer cannot keep up it will drop frames from the video stream. Real time audio compression is usually possible. However, if you choose to compress the video after sampling, you may prefer to defer audio compression also.

### *Compress after capture*

If you select this option, the data is recorded raw to a temporary file. Make sure you have considered how large this file can get. Recording 640x480 images at 30 fps can easily use more than 1 GB a minute. When sampling has stopped and you choose to save the data file, the temporary file is added to a queue of files to be compressed. The compression task runs in the background, so that the program can continue capturing data. Spike2 will not be able to access the compressed files until the compression task has finished.

### *Disable compression*

This option gives you the fastest sampling and access to your files, but at the cost of the largest files.

### *Codec*

This option lets you choose the codec that is used to compress your video or audio data stream. If you get an error message of the form, "*Sorry, a DirectShow error occurred. Action: Render the output stream. Change or disable video/audio compressor. The error encountered was: No combination of intermediate filters could be found to make the connection*", this means that you have chosen a codec that either required an input format or that generate an output format that was not compatible with the data stream. You must choose a different one.

## **Frame timing**

AVI files hold a list of frames (which can be empty of data if the frame was dropped) at a constant time interval. The frame rate is the number of frames per second and this must be set accurately if Spike2 is to be able to scroll a video display to correspond with a time in a data file. When capture of a video file ends the frame rate is set in the AVI file and you can choose to control how the frame timing is recorded. These options were added at Spike2 7.09.

### *Frame rate adjustment*

There are three options: None, Frames/time and Fixed value.

- |             |                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| None        | Accept the frame rate generated by Windows, which will usually be the rate that the source hardware claims to generate.                                                                                                                                                                                                                                                                                                        |
| Frames/time | Count the number of frames stored into the AVI file and divide this by the sample time in seconds. If the video really does run at a constant rate, this should generate a video file that lines up in Spike2.                                                                                                                                                                                                                 |
| Fixed value | Set the frame rate to a user-defined value. If you are triggering the frames with a clock that is known synchronised to Spike2, such as a clock generated by the output sequence Clock output, then you will know the exact clock period, so use this mode. We know of one triggerable camera that seems to guess a rate based on the image size when set to triggered mode. Setting the known rate works around this problem. |

### *Multimedia time offset to align with Spike2*

There will almost always be a time offset between the time that Spike2 starts running and the time that the camera records its first frame. This can be positive (if the first frame is late) or negative (if the system buffers up a few frames, ready for you to say go). If your device tends to have the same offset each time, you can set it here, and we save it in the AVI file. Spike2 will apply this offset when you review the file. The time is saved to millisecond precision.

This field is experimental and we store the information in a supposedly unused section of the AVI file header. AVI files not generated by s2video will always have this value set to 0. You can also adjust this value in Spike2 to line up the video exactly, assuming that the frame rate is correct.

### *How to check video timing*

If you set the following sequencer script:

```
set 1,1,0
go: digout [00000000]
 delay s(5)-2
```

```
digout [.....11]
delay s(5)-2,go
```

This causes the lower two digital output bits to change state every 5 seconds. This causes the front panel LEDs on Micro1401 and Power1401 devices to switch on and off. If you connect digital output 0 to event 0 input on the front panel, then sample the input as a Level event channel, you can record the precise timing of the digital output changing state. If you point your video camera at the front panel, you can now record a video that should show the front panel LEDs changing at exactly the times when the Level event channel changes. If the frame rate is exactly right, you are more likely to see that the changes are late by a more or less constant time (remember that no times can be more accurate than the frame rate). If this delay stays constant over a range of video files, you can set this delay value as the *First frame delay*.

## Working around problems

Windows allows manufacturers to generate drivers for a huge range of devices. The interface specifications are complex and not all devices implement all aspects of the specifications perfectly. Because of this, we have added workarounds for problems that occasionally occur.

### *Disable drift compensation*

This is a workaround for some DirectShow audio device drivers that crash or generate errors when we try to match the multimedia time base to the Spike2 time. If you have problems of this nature, try checking this box (but the multimedia time and Spike2 time will drift apart). You can read more about drift compensation here. If you set *Frames/time* in the *Frame rate adjustment* there can sometimes be an advantage to checking this box as you are going to rescale the timebase using the frame rate.

### *Disable video (and audio) during Properties dialog*

We normally leave the multimedia capture running when you open the device properties dialogs. If you have Preview enabled, this allows you to see (and hear) the effect of any changes you make. However, some devices disable aspects of their Properties dialog if the device is being used. Check this box to stop running the device while the Properties dialogs are open.

## Storage of settings

The program stores its configuration settings in the registry in the key:

```
HKEY_CURRENT_USER\Software\CED\Spike2 Video
```

## Codecs

You can choose to compress the video and audio data so that it occupies less disk space. This is done using plug in modules called codecs. A codec is in two parts: a coder and a decoder, hence codec. The coder changes the input signal in some way and the decoder reverses the process. Not all codecs are compressors; some may just change the format of the data to suit a particular purpose or to make the content secure. We are only interested in codecs that can make the data significantly smaller and that can do it fast enough to be useful.

Some codecs are lossless, that is, the decoded data exactly matches the encoded data; these tend to produce only small amounts of compression in real world data. Most codecs are lossy; the decoded result is not exactly the same as the original. However, they are designed to lose information in parts of the signal that we are less sensitive to, so often the result may look and sound close enough to the original not to matter.

Different codecs are also designed for different purposes. For example, the Microsoft RLE video codec is designed to compress images with areas of identical colour, so it does a very good job on cartoons but makes a terrible mess of real-world images.

When you installed DirectX, a set of standard codecs was added to your system. It is likely that when you installed your video camera or web cam, the installation software added further codecs. The **Audio codec** and **Video codec** fields let you select codecs that advertise themselves on your system as supporting audio or video. Not all of these are useful, or will even work with `s2video`.

The codecs listed are those I found pre-installed on my quad core Windows 7-64 system in November 2011, plus DivX, which I installed as I knew of it. There are, no doubt, many other excellent codecs in existence, their omission from the lists is just an admission of my ignorance. This is not a scientific test but it give a flavour of what you can expect without doing any specialised codec set up to tune them for the data.

I also experimented with codecs available from various free codec packs on the web. I did not find one I could

recommend; they tended to be slow at encoding or were rejected for other reasons, for example putting up dialog boxes when they were activated or causing crashes. It is likely that such codecs can be used if you take the time to find out how to configure them, but I was looking for easily available codecs that "just worked".

### Video codecs

To give you some ideas of which video codecs you could consider, the following table gives number of kB per second required to record of an image of me moving about in front of a web camera for 10 seconds. The camera was set to produce 320x240 pixel images in colour at 30 frames per second. I converted the images on-the-fly; if the codex was slow, it dropped frames to keep up.

| Codec                   | kB/sec | Factor | Comments                                                 |
|-------------------------|--------|--------|----------------------------------------------------------|
| None                    | 6900   | 1      | Uncompressed video                                       |
| DivX                    | 119    | 58     | Good image, fast encode, uses multiple CPU cores         |
| Microsoft Video 1       | 163    | 42     | Blocky image                                             |
| Cinepak Codec by Radius | 230    | 30     | Blocky image, dropped frames                             |
| Microsoft RLE           | 1554   | 4.4    | Bad image; the codec is not meant for this type of data. |
| Intel IYUV codec        | 2188   | 3.2    | Good image                                               |
| MJPEG Compressor        | 2584   | 2.7    | Good image                                               |
| DV Video Encoder        | 3740   | 1.8    | Good image                                               |

If you want 640x480 images, the data rate could be 4 times higher. If you want HD images (1920x1080), the data rates may be 27 times higher and you will likely need specialised hardware compressors.

### Audio codecs

There is a wide range of audio codecs to choose from. Some are optimised for a particular task, such as compressing speech. The table shows the result of compressing me speaking the numbers 1 to 14 in approximately 11 seconds using stereo 16-bit sound at 44.1 kHz (which is the minimum setting allowed by my sound card).

| Codec                 | kB/sec | Factor | Comments                                                 |
|-----------------------|--------|--------|----------------------------------------------------------|
| None                  | 180    | 1      | Uncompressed audio                                       |
| PCM                   | 180    | 1      | This is uncompressed audio in PCM format                 |
| WM Speech Encoder DMO | 8.3    | 22     | Muffled, but understandable. Use only for speech!        |
| GSM 6.10              | 15     | 12     | OK                                                       |
| WMAudio Encoder DMO   | 47     | 3,8    | OK. Artefacts (probably caused by clock time adjustment) |
| IMA ADPCM             | 55     | 3.3    | OK                                                       |
| Microsoft ADPCM       | 55     | 3.3    | OK                                                       |
| CCITT A-Law           | 97     | 1.9    | OK                                                       |
| CCITT u-Law           | 99     | 1.8    | OK                                                       |

The uncompressed file rate of 180 kB/s is the 44.1 kHz rate times 4 bytes per sample plus overhead. You can reduce the audio data rate significantly by using 8-bit mono sound input (if your sound hardware allows this), which gets you down to 45 kB/s or so without a codec. If you can tolerate a lower sampling rate, this will also reduce the rate. Remember that you are trading off disk space, which mainly means the time taken to write to disk, against the time taken to encode the data. For example, if your sound hardware allows you to sample mono 8-bit data at 20 kHz and you are also sampling video, you may decide that the audio is the least of your problems and it is not worth the time to compress it.

If the exact synchronisation of the sound with other spike channels is important, record the sound as a Waveform channel. It is possible to switch recording of specific channels on and off to save disk space.

## Disable drift compensation

Time in a Spike2 data file is measured based on a crystal-controlled clock inside your 1401 interface. Time in a multimedia file is based on either the computer system clock or on a clock in some multimedia hardware. The 1401 interfaces use crystals that are accurate to 50 parts per million over a 0 to 70 degrees centigrade range, this is a worst case gain or loss of 4.3 seconds a day; most units will be better than that. Assuming that the clock in your computer is of similar accuracy to the 1401 (and it could be much worse), there is a possibility of the multimedia recording and the 1401 disagreeing about time by as much as 8.6 seconds per 24 hours.

To avoid problems with time drift, Spike2 keeps track of the time differences between 1401 time and computer time and notifies the `s2video` application of any drift. This drift can then be compensated for to make sure that the times in the multimedia file remain locked to those in the Spike2 data file.

Unfortunately, we have come across audio device drivers that crash when we try to adjust the time. Usually you can fix this by getting the most up-to-date driver for the audio device, but if this is not possible you can work around this problem by disabling the time drift compensation.

Even with the time drift compensation in place, there will be a fixed time shift between the Spike2 data and the multimedia file due to the time it takes the system to process the video and audio data. This fixed shift depends on the hardware in use and is usually less than 300 milliseconds.

Do not confuse the problems of time drift (which are usually small) with gross effects due to video devices claiming to run at one frame rate when they actually run at a different one. For example, I have a cheap USB camera that claims to run at 30 frames per second, but actually runs at 31.8 frames per second. See the Configuration dialog to compensate for this kind of problem.

## View menu

The view menu contains four items:

### Toolbar

The toolbar holds short-cuts to preview the video image, to reduce the frame rate to the value set in the Settings menu **Set Slow Frame Rate** and to open the configuration menu. You can choose to show or hide it with this command.

### Status bar

You can choose to show or hide the status bar at the bottom of the window. The status bar gives you feedback about your menu selections and information about recording.

### Always on top

Tick this menu item to keep the `s2video` application on top of other windows.

### Preview

When a video camera is selected this shows or hides the video preview window.

## File menu

This contains a single option, **Exit**, which will close the application unless you are recording or compressing data files.

## Recording data

The `s2video` application acts as a slave to Spike2. If Spike2 is not running, you can configure the program, but it will not record data. Once a copy of Spike2 is located, `s2video` registers itself as a "listener" application. Spike2 will now tell it about all recording activities: opening a new temporary data file ready to record, starting recording, aborting or resetting sampling, stopping sampling, saving the temporary data file as a permanent file or throwing it away.

The `s2video` application does exactly the same tasks with its own multimedia file. There is one extra step required if you have chosen the **Compress after Capture** option in the configuration dialog. Instead of moving the temporary file to its final destination, it must be compressed. Compression can take a long time, and Spike2 might want to sample another file immediately. To avoid holding up Spike2, the temporary file is added to a list of files that need compression and the compression is run as a background task.

Compressing a file can take a long time, so while it happens `s2video` displays a dialog showing how far through the task the compressor has got for the current file. If there are multiple files waiting in the queue, the number of files left to process is also displayed.

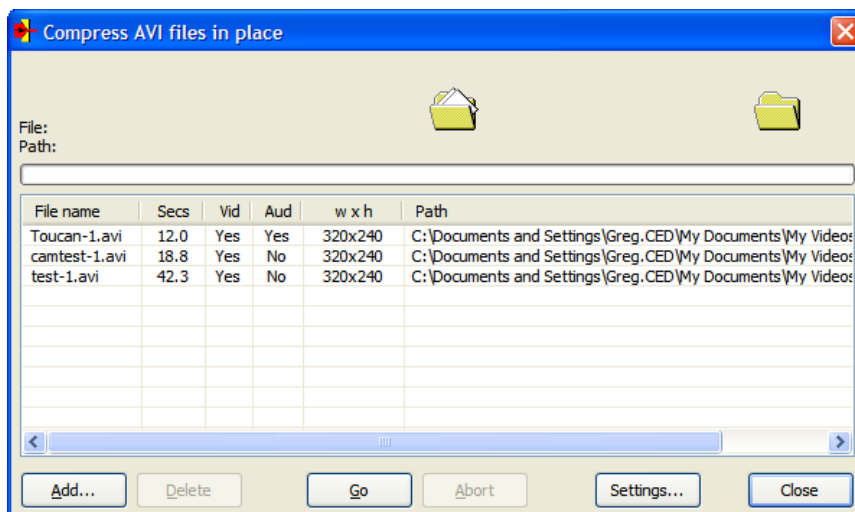
In long recording sessions, most of the time you may need one video frame every few seconds, but there are short periods where you need the full frame rate. There is a Spike2 script command `MMRate()` that you can use to request a new frame rate. This command can only generate rates up to the maximum frame rate set for the camera.

When recording data under script control, you must use `FileSaveAs(name$, -1)` to save the data file and multimedia files. `FileSaveAs(name$, 0)` exports the data file and does not save any multimedia files.

If you get video stream timing problems, check that any frame rates set in the Video Device Properties match those in the Video Capture dialog. Beware of Auto options that may set unexpectedly high rates; it is usually better to set the rates manually, so that you know what they are. If the problems persist, set a lower frame rate or a smaller image size; the most common problem is that the system cannot process data fast enough.

## The avicomp application

The `avicomp` application can be used to compress `.avi` files generated by `s2video` that were not compressed during data capture. To run the application, double-click the file `avicomp.exe` that is in the same place as the Spike2 application.



*avicomp main window*

The application main window is in three sections. The upper section holds the name and path of a file that is being compressed, with a progress bar that indicates how much of the file has been processed. The middle section holds a list of files that are to be processed and also displays the length of the file in seconds, if it holds video and audio and the size of any video image. The bottom section holds a list of buttons that you can use to control the program. The buttons are:

## Add

This button opens a standard file selection dialog in which you can select multiple .avi files. To be added to the list, each file must contain an AVI file (the name need not have the .avi extension, but all files created by s2video do have this extension). The files must also be readable and writable. You can always add a file, even when compressing. You cannot add a file that is already in the list. If there is a problem adding a file a message box opens to give you a clue as to what went wrong.

## Delete

This button is enabled when there is a selection in the list. The command removes the selected files from the list.

## Go

Start compressing the files. Each file is removed from the list as it is compressed and the file name, path and compression progress are displayed in the upper part of the display. Files are compressed to a temporary file in the same folder, then if the compression is successful, the original file is deleted and the temporary file is renamed to the original name. If you want to preserve the original file, you must save a copy first.

## Abort

Click this button to stop compressing the current file. Any temporary file is deleted and the file will be put back into the list. The compression process will stop.

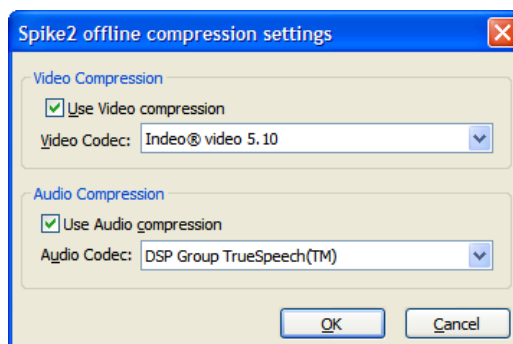
## Settings...

This command opens a dialog where you can choose the video and audio compression filters to apply to your file. You can change settings, even while processing files. The settings used will be those in effect when each file starts to be processed.

## Close

This button is enabled when you are not compressing any data. It saves the list of file names in the window (up to a maximum of 100 files) in the registry, then closes the application. The list of files will be reloaded the next time you run the avicomp application.

## avicomp compression settings



*Offline compression settings*

From this dialog you can select the video and audio codec to apply to your data stream. The dialog will not let you exit unless you enable at least one compressor.

## AVIComp errors on file add

Adding files to the list can fail for the following reasons:

### **Already open or a disk error**

The file is open in another application or there was another problem that cause the attempt to open the file to fail.

### **Already in the list**

This is self-explanatory. It is a waste of time to attempt to compress the same file twice and would (at best) result in a loss of quality as the file would be uncompressed, then compressed.

### **Not an AVI file**

Again, self-explanatory. The system failed to recognise the file as an AVI container - probably because you have selected the wrong file.

### **An AVI file but ran out of memory**

The system recognised the file format, but ran out of memory trying to extract information about the contents. This is most unusual on a modern system and may mean that the file is corrupted.

### **An AVI file but no driver**

You may also so: (REGDB\_E\_CLASSNOTREG: missing registry entries?) This means that the file was recognised, but when Windows looked up the type of a data stream in the file it could not locate it in the system registry. You can get this error legitimately if the codecs needed for a file do not exist on the system. You can also get this when the registry entries in [HKEY\_CLASSES\_ROOT\AVIfile] are missing.



## **22: Technical support**

# Technical support

## How to contact CED

### Technical support

There is a bulletin board at <http://www.ced.co.uk/phpBB3/> where you can post questions and get responses from CED and also from other Spike2 users. It also enables others to learn from your questions and the responses you get.

We also have software and hardware technical support desks that are available during UK working hours (and somewhat beyond). You can call us by telephone or send us a message by FAX, but our preferred medium is email as you can send example files and scripts as enclosures (even from within Spike2 with the File menu Send Mail option). If your question is not absolutely specific to your own circumstances, you might care to pose it on our Bulletin Board system so that others can benefit from the answer (you may even get a response from other CED users).

USA and Canada: 1 800 345 7794  
World-wide: +44 1223 420186  
FAX: +44 1223 420488  
email: [softhelp@ced.co.uk](mailto:softhelp@ced.co.uk)  
email: [hardhelp@ced.co.uk](mailto:hardhelp@ced.co.uk)  
BBoard: <http://www.ced.co.uk/phpBB3/>

You can also find up to date information, example scripts, and Spike2 down-loadable updates on our **Web site** at <http://www.ced.co.uk>

If you send us an enclosure, please have mercy... don't send us gigabytes. We have a medium speed link so a few megabytes is fine, but please contact us first if you need to send more than this. Most problems that need a file can be illustrated with a file section (use the File menu Export option to chop out a small piece). If you need to send us a large file, consider using a CD or DVD and the mail.

Before you contact us please make sure you have checked that the information you need is not in the manuals or the on-line help. The on-line help is more up-to-date than the manuals and the Index is always worth a try; remember to try a few synonyms when searching for information.

Please tell us the serial number of your copy of Spike2 (this can be found in About Spike2 in the Help menu; to make it easy the Copy button will place the dialog information on the clipboard as text), the Spike2 version number, the operating system version, and any hardware-related information that might be relevant to the problem. If you have a crashing problem, please read the information about Dr Watson (below).

If your problem is sampling related, please send us information about your 1401. The easiest way to do this is to run the Try1401 test program and use the File menu 1401 info... command and email us the results. A copy of the sampling configuration (.s2c file) will allow us to reproduce your sampling setup. Please remember that we will also need any other files that make up your sampling configuration (output sequences, arbitrary waveforms, scripts). If you have a complex arrangement of folders, scripts and sequences, please consider simplifying things - you will get a fix much more quickly if you send us the simplest arrangement that causes a problem.

### Using Dr Watson to report crash information

If Spike2 crashed out with a system error (General Protection fault, or the like), please send us all the information that the system gives you as often we can pinpoint the offending line in the code given the register dump. The best way to get the information is by using the DrWtsn32 program. If this program is enabled, it writes a log file called `DrWtsn32.log` that describes what happened during a crash. This file is written to the folder:

```
%systemdrive%\Documents and Settings\All Users\Application Data\Microsoft\Dr Watson
```

unless you have explicitly changed this by running the `DrWtsn32.exe` program and modified the save path. `%systemdrive%` is the drive where your operating system is installed, usually C:. If you are not getting any log files written, open a command prompt and use the command:

---

DrWtsn32 -i

to enable it. You can also run the program by typing `DrWtsn32` at the command prompt to configure other options. The log file is a text file that you can open and read. Normally, each crash is appended to the end of the file, so if your log file is huge, you may wish to only send us the most recent information from the end. Sending us this log file plus a description of what you were doing often makes finding and fixing problems much quicker. If the problem is easy to reproduce, please send us all the steps required to reproduce it. If it only happens very occasionally with no obvious cause, the Dr Watson log file is essential. We give a very high priority to finding and fixing crashing problems; we will generate a fix for the next release and may even be able to send you a fix or a work-around.

### Crash on startup

If Spike2 crashes during startup, there is a section of the startup code beyond our control where it reads in control bar positions, and if the registry becomes corrupt for any reason, this can stop the program starting. You can delete some registry entries to fix this, see the Script language `Profile()` command to read more.

### Sales related enquiries

Write, telephone, FAX or email us at:

Cambridge Electronic Design Ltd  
Science Park  
Milton Road  
Cambridge  
CB4 0FE  
UK

Telephone: +44 1223 420186

FAX: +44 1223 420488

email: [sales@ced.co.uk](mailto:sales@ced.co.uk)

If you want price list or product information, you could try our World Wide Web page: <http://www.ced.co.uk>

## Recent Spike2 revision history

The following changes, additions and fixes have been made to Spike2 since version 8.00. You can also get update information, downloads and script examples from our web site. The revision history lists items under the following headings:

- New** Extra features in the program or improvements in operation.
- Fixes** Repairs of broken features (bugs).
- Changes** An alteration to Spike2 (which may also be listed in New or Fixes) that may require you to change existing scripts or program operations. We try to minimise these.

The **Known issues** section lists problems for which there is no fix currently available.

## Revision 8.02

These are the new features, fixes and changes since version 8.01c.

### New

1. You can now choose to be warned if closing a data file would lose data from a memory buffer channel. By default, we warn you when you close a file interactively, but not when you close it from a script; you can change this in the Preferences.
2. When debugging a script, the text caret moves to the start of each statement rather than the start of the line containing the statement; this is helpful when a script has multiple statements on one line.
3. Right-click on the title bar of a view to get the option to copy the full path to any associated file to the clipboard.
4. In the clustering window, the INTH display window now holds a vertical cursor set to the minimum

interval. You can modify the minimum interval by dragging the cursor.

5. The Plexon importer now recognises off-line modified PLX data files.
6. The Alpha Omega importer now identifies all types of Event and marker data in the MPX file format.
7. The Version 5 DSI importer is now very much faster for large datasets (we have reports of 100 times faster). You must install the 32-bit version of Spike2 to use this importer as the interface library supplied by DSI is 32-bit only.
8. The Axon Instruments importer is no longer limited to files with less than 65536 data blocks. You must install the 32-bit version of Spike2 to use this importer as the interface library supplied by Axon is 32-bit only.
9. The ASCII importer now stores configuration information in XML files only. It has been recoded for this release to make it more robust. See the interactive help supplied with the importer for more details.

## Fixes

1. Script errors in a function linked to a toolbar, dialog or a dialog button did not indicate the script line that caused them.
2. Script errors from built-in commands that were not related to a specific argument, for example calling `DlgShow()` with too few arguments, reported an error number rather than a more helpful message.
3. During File Export from a script, a progress bar appeared, but the script was not notified if the user cancelled the export. When run interactively, no progress bar appeared. You now get a progress bar for interactive export and none for script export. In the future we may allow script-driven export to request a progress bar.
4. When recording your actions, clicking on a text-based view did not always record a `FrontView()` command or add `ViewFind("view title...");` to the start of the script.
5. Marker filter OR mode was testing marker code `n` against mask `n` instead of testing them all against the first marker mask.
6. The documentation incorrectly indicated the direction of the NDR and NDRL digital output pulses on modern 1401s.
7. In a 64-bit smrx file, if you deleted a channel and reused it, subsequent reads from the channel could fail, usually resulting in gaps in the display. The data was written correctly, but the index table could have incorrect lookup information. Files in this state are repairable in `S64Fix`.
8. File names for 32-bit data files generated through the Sampling Configuration dialog Automation page always had `.smrx` file extensions, which prevented Spike2 from opening them. You can open such files by changing the extension to `.smr`.
9. Renumbering cursors or using `CursorRenumber()` with multiple cursors on the same pixel could leave a ghost cursor behind.
10. The Graphical sequence editor can now set random Poisson statistic delays that are very long compared to the sequencer step period. Previously, the longest delay that could be set with a 0.01 ms step period was a second or so; it is now many minutes.
11. If you saved a result view with a Raster display to a data file, it was possible to get an incorrectly scaled Y axis the next time you opened it.

## Changes

1. By default you are warned about losing memory channel data when you close a file. You can change this to the old behaviour (no warnings) in the Preferences.

## Revision 8.01

These are the new features, fixes and changes since version 8.00a.

### New

1. Mult-trace WaveMark data (Stereotrode and Tetrode) can now have non-sequential ports in the 1401. The

---

`SampleWaveMark()` and `SampleChanInfo()` script commands are extended to support this.

2. The Sample bar and the Script bar now have a context menu (right-click) command to remove a button.
3. The current Talker interface specification is now version 2 and script language users can now communicate with talkers that support this with the new `TalkerSendStr()` and `TalkerReadStr()` commands.
4. Talkers can request sampling to start and stop and can request that a named script runs. This can allow remote control of Spike2 data capture.
5. In the script editor, you can navigate to a user-defined `Func` or `Proc` (even if the code for it is in an included file) by right-clicking on the name.
6. In the script editor, if you right-click on a built-in function or script keyword, the context menu includes the command `Help for...` and selecting it will open the best match in the help file to that item. This is equivalent to using the `F1` key with the text caret on an item.
7. In the script editor, the auto-completion list now contains user-defined `Func` and `Proc` items from included files. For this to work you must check the `Included files` box in the auto-completion dialog.
8. In the script editor, hover the mouse over a user-defined or built-in `Proc` or `Func` name to show a tool tip for that item. You can set your own tool tips for user-defined functions.
9. In the script editor, call tips (tips that appear when you type the '(' after a function name) now track your typing and offer additional tips if you type more functions names known to the tip system as function arguments.
10. The `MarkMask()` script command is extended to report if a marker filter is active.
11. The `PlayWaveCopy()` script command can now move data in either direction.
12. The `Modified()` script command is extended in a Time view to force data through to the disk, equivalent to the Automation tab timed Flush to disk in the Sampling Configuration dialog. There is more information about data buffering in Spike2.
13. MATLAB data export in a time view is now faster in most cases and waveform export has new options to ignore gaps and to force the same number of points when exporting multiple channels sampled at the same rate.
14. The Active Cursor dialog now tracks user edits and disables OK if a field is incorrect rather than displaying a warning message box. Error messages now appear as part of the dialog.
15. Importers: The Axon, MC\_Rack and Biopac importers are now linked with the latest libraries (but still only available in 32-bit builds of Spike2 as there are no 64-bit libraries available). You can select the recording session in the NewBehavior (Neurologger) importer.

## Fixes

1. It was possible to drag the Y axis of a Spike shape dialog (except in online setup) to values beyond the range of the data. The data was correctly limited, but the axis was not.
2. When sampling tetrode data (4 traces), the on-line display of non-triggered data in the Spike shape dialog displayed traces 0,2,2,3. This did not affect recorded data.
3. Recording of `MeasureToChan()` and `MeasureToXY()` omitted the `width` argument in threshold iteration modes.
4. Level events channels written to a 32-bit smr file were inverted.
5. If level event data recorded to a 64-bit smrx file contained events at duplicated times, the mechanism to remove these events could fail, and it was possible for two events at the same time to be scheduled for writing to disk; this was detected as an error and sampling would stop.
6. Using `Modified(0,0)` during sampling with a Level event channel could cause sampling to stop with Error -23.
7. Exporting more than 20000 event times to a MATLAB file could crash Spike2.
8. It was possible to get the Active Cursor dialog confused by setting illegal values, then changing the cursor number. You are now not allowed to change the cursor number if the dialog is in an illegal state.
9. The script compiler did not flag a comparison between a number and a string as an error. For example: `if`

```
3 = "three" then halt endif;
```

10. Importers: The y scaling has been corrected in both the Alpha Omega and Neuralynx importers. The ASCII (text) importer could get the times of TextMark data confused. The CFS importer now gets the times from Marker channels correctly.

#### **8.01a**

11. If you created an XY data view, then used `XYKey(1,0)` to hide any key, the key would appear and adding data to the view could cause Spike2 to crash.
12. The end time (`eT`) argument of the `MarkSet()` script command was not included in the time range. If there was a marker at time `t`, `MarkSet(chan%, t, t, ...)` did not change the marker.
13. If the Sample control toolbar was hidden when Spike2 started, the toolbar buttons could be displayed too high in the bar.

#### **8.01b**

14. Changing the output sequence interactively or from a script during sampling did not work.
15. In a user-defined dialog with a change call-back function, if you changed the active field by clicking, the change function was called for `DlgReal()` and `DlgInteger()` fields even if they had not changed.

#### **8.01c**

16. Backwards searches through WaveMark channels treated as waveforms did not work in 64-bit files. In 32-bit files it was possible to get a result that was after the start time.
17. Trough amplitude measurements for events drawn in instantaneous and mean frequency and rate mode and for RealMark data were incorrect.
18. In the FIR filter dialog, if you selected the All Pass or All Stop filters, the displayed waveform did not always update.
19. It was possible for multiple sets of filters to appear in the FIR filter dialog. If you have this problem you may wish to delete your filter bank file.

### **Changes**

1. In the Graphical sequence editor we now refer to sections as **Section A** to **Section Z**, not **Key A** to **Key Z**.
2. Spike2 will not sample with out of date 1401 device drivers. Spike2 installations always copy the latest drivers, so this should not be an issue.

#### **8.01b**

3. When Y axis labels are set horizontal, the y axis units display unless the axis is less than 2 characters high (was 4 characters).

#### **8.01c**

4. Peak and trough measurements now take into account the baseline when detecting the maximum and minimum values. Previously, the absolute maximum or minimum was used and then adjusted for the baseline.

## **Revision 8.00**

These are the new features in version 8 compared to 7.12.

### **New**

1. Data is now stored in a new filing system that allows files of pretty much unlimited size and duration; the old system is still supported so old files will read and can be modified.
2. With the new filing system, sampling times are no longer limited to 2 billion clock ticks; at a clock tick of 1 microsecond you could sample for many years.
3. You can choose which format to sample to (32-bit or 64-bit) in the Sampling Configuration dialog.
4. With 64-bit files, triggered and timed waveforms save exactly the required data; 32-bit files always save complete disk blocks which usually saves more data than required.

5. If you run on a 64-bit operating system we now install a 64-bit version of the program. This runs faster than the 32-bit version and allows the program access to more memory.
6. New S64Fix utility program to recover data from damaged 64-bit data files (similar to the SonFix program for 32-bit files).
7. You can now set horizontal spike amplitude limit cursors when sampling with a Micro1401 mk II (to match the other supported 1401 types). These set the upper and lower limits for acceptable spikes.
8. The sonogram display copes better with gaps in the data and now has an optional key to indicate intensity.
9. The new ChanKey() script command supports channel keys (currently for sonograms and XY views)
10. If you import waveform data into a RealMark memory buffer, the peak/trough/level value associated with the import is saved in the first value associated with each marker. You can also choose to import both Peak and Trough times or both Rising and Falling times in one operation. The MemImport() script command has been extended to match.
11. The output sequencer text editor has Keys and Labels controls to navigate to any key press or label in the sequence.
12. Script bar, sample bar: right-clicking a button opens a context menu from which you can open the script or configuration.
13. Sample bar: the button context menu allows you to sample immediately and to control the write enable state.
14. The script language allows you to define default values for proc and func arguments.
15. Script integer values are now 64 bits (they were 32 bits in previous versions). This allows ChanData() on an event-based channel to return 64-bit times as integers.
16. The output sequencer has a new instruction TICK0 and TICKS has been extended to cope better with sample times that are too large to fit in the sequencer 32-bit variables. There is a new script command, SampleSeqTick0().
17. The output sequencer has new sequencer expression functions sTk64h() and sTk64l() to convert times in seconds to 64-bit tick counts.
18. The BReadSize() script command now accepts -n when reading a string. This stops reading at a null (ASCII code zero) character or after n characters.
19. The BWriteSize() and BReadSize() script commands now accept 8 as a size for integers (for 64-bit data). The size used by BRead() and BWrite() for integers is 4 (32-bit) for backwards compatibility.
20. The BSeek() script command can now seek to positions with a 64-bit offset into the file. Spike2 version 7 was limited to 32-bit offsets.
21. The FileSaveAs() script command now adds a file extension if none is supplied. When used with a multimedia window to save the current frame as a bitmap you can now set a blank name or a name with wildcards to prompt the user for a file name (as for other window types).
22. The FileSaveAs() script command old flags% argument is renamed to expt% and the yes% argument is renamed to flags%.
23. The maximum size of a user-defined dialog is now limited to the size of the primary monitor screen. Previously it was set to a smaller, arbitrary limit. The previous arbitrary limit is used if the primary screen size is less than the old limit, for backwards compatibility.
- 24.DlgCreate() accepts negative wide and high values to limit the dialog size if it exceeds the set size.
25. The Y Axis Range dialog has a spinner on the channel selector.
26. There are improvements to the data import system to allow longer time ranges and larger files and to give more information and control during interactive file importation.
27. You can define Alt key combinations to activate buttons in the Script bar and Sample bar.
28. The newDlgFont() script command gives you more control over the font used in user-defined dialogs.
29. The SampleWrite() script command is extended to mark time ranges for saving.
30. The MATLAB file export system can write files larger than 4 GB if you select the version 7.3 format. Beware that 32-bit versions of MATLAB cannot cope with huge data files.

31. If you generate stereotrode or tetrode data from waveform channels with different channel units you can now choose to continue, assuming the units and scaling of the first selected channel.
32. The `ArrStats()` script command calculates the mean, variance, skewness and kurtosis of an array.
33. The `MATTrace()` script command calculates the trace of a matrix.
34. Version 8.00a added the `ArrHist()` script command.

### Fixes (in 8.00a)

1. Backward searches of waveform channels with a slope channel process could cause Spike2 to hang.
2. Reusing a deleted channel in a `.smrx` (64-bit) file with a different channel type could cause data lookup failure.
3. Calling `ChanList()` with no time, result or XY view reported a wrong view error rather than returning 0, as documented.

### Changes

4. The *1401plus* and the original *micro1401* are no longer supported for sampling. If you need to sample with these, use Spike2 version 7 (also included on the distribution CD).
5. To sample data, your 1401 must have up to date firmware. If the firmware is out of date you will be prompted to update it when you try to sample data. You can check if you are up to date in the Help menu About Spike2 command. Starting version 8 with modern firmware allows us to streamline data capture, avoiding supporting old and inefficient data transfer protocols.
6. The Script menu **Convert DOS script** command to convert scripts from the MS-DOS version of Spike2 has been removed. This is still present in Spike2 version 7, if needed.
7. The `yes%` argument of the `FileSaveAs()` script command is now a set of flags. If a version 7 script uses an argument value that is not 1 or 0 you may not get identical results in version 8. Replace the argument with `(argument) != 0` and it should work exactly as the old version did.
8. The `ScriptBar()` and `SampleBar()` commands remove trailing "|" characters when returning the string associated with a button. This is unlikely to cause problems in existing scripts.
9. The default Working Set size has been increased.
10. We no longer provide printed, hand-crafted manuals (most users did not make use of them and the on-line help had live links and held more information, further, it took us considerable effort to maintain the manual in addition to the help file). Instead, we include a PDF of a printable manual that is generated from the on-line help.
11. We no longer install the `SonCols` or `SonInfo` shell extensions; this is because of changes Microsoft have made to shell extensions.
12. The file importers now create 64-bit `.smrx` files, not 32-bit `.smr` files. Scripts that import files, then further process them may need modification to open a `.smrx` file rather than a `.smr` file.
13. Both `FileSaveAs()` and `FileNew()` have modified the meaning of the `big%` argument to cope with 64-bit data files. If you used this argument in a script and want to create a 64-bit file you should change the argument to 2.
14. The `SampleBigFile()` script command has modified the meaning of its optional argument. If you used this argument in a script and want to sample to 64-bit files you will need to change it to 2.

## Frequently asked questions

The following questions have been asked several times over the history of Spike2:



---

## I've set a sampling configuration. How do I sample data with Spike2?

Make sure you have a 1401 interface connected to your computer and switched on. You must also have installed the appropriate 1401 device driver (the Spike2 installation will have done this unless you told it not to).

Use the File menu New command and select a data file. This opens a new, empty data file. You can start sampling either by using the Sample menu Start sampling command, or clicking the Sample button in the Sampling control dialog box.

You can also start sampling from the Sampling configuration dialog by clicking the Run now button.

If you have several sampling configurations that you use frequently, load them into the Sample Bar. You can then load the sampling configurations with one mouse click and start sampling with a second or you can set the sample bar to start the configuration running immediately by right-clicking on the sample bar button and using the Immediate Start option.

## Spike2 beeps when I try to open a new file for sampling or exit

If you have a new data file, either sampling or about to sample, you cannot open another or exit from Spike2. If you try to open a new file or exit from the Spike2 menu, the program beeps to remind you that the data file is open. A script will give an error if you try to open a new file.

This problem usually happens when you have been working with a script that samples data. Such a script may open a data file "hidden", so it is not obvious that there is a data file. Use the Window menu Show command to find any hidden windows or the Window menu Windows command to list all windows..

## What is the Log window?

The log window is a useful scratch text area that is always available. It behaves exactly like any other text window except that deleting the Log window only hides it; use the Window menu Show command if it is hidden or type `Alt+W` then `l`.

You can type text into the window, or text may be added by a script or when an error occurs or to help us to diagnose problems. You can also save the contents of the log view as a text file. You can limit the number of text lines in the Log view from the Edit Menu Preferences General tab.

The log window is the destination for output from the `PrintLog()`, `DebugList()` and `DebugOpts()` script commands, for any output from the Evaluate toolbar and is also used to dump some information if a script hits an error. You can also write the current sampling configuration as text to the Log window from the Sampling Configuration Channels tab.

Script uses can get the handle of this window with the `LogHandle()` script command.

## When I sample data, where is it saved during sampling?

Unless you have set a file name in the Automation Tab of the Sampling configuration dialog, Spike2 saves new data in a file called `Data` plus a sequence number in the path (folder) set by the Edit menu Preferences->Sampling command. If no path is set in the Preferences, the data goes to the folder set for data. Where this is exactly depends on the options you chose when installing the system. If you chose to install Spike2 into the system Program Files folder, the file will be typically found in:

```
c:\ProgramData\CED\Spike8\
```

One way to discover exactly where your files are being saved is to:

1. Sample a new file for a second or so, stop sampling but do NOT save the file.
2. Go to the Windows Start menu and in "Search Programs and Files" type `regedit`. Windows should offer you `regedit` as a runnable program, so run it (despite the dire warnings from the operating system).

3. Open: HKEY\_CURRENT\_USER\Software\CED\Spike2\Recover
4. You should see that the key Last is set to the path to your temporary data file and Type is set to smr or smrx, depending on the file type.

Once you save the file, the registry entry is wiped. If Spike2 is interrupted during sampling by a power or system failure, the next time you run Spike2 it will detect that the last sampling session did not terminate as expected. You will be asked if you wish to attempt to recover the data.

We recommend that you set a path in the preferences so that you can control where temporary data is saved. The data is stored as a Spike2 data file without the file extension. When you close the file or use the Save command, you are prompted to supply a file name and the original file is renamed if the new name is on the same drive (volume), or copied if the new name is on a different drive. As files can be large and take a long time to copy, you should try to make sure the same drive is used for recording as for saving. On the other hand, if your saving drive is across a network you really do not want to record to it!

## Do I have to use scripts to use Spike2?

No, you do not. Many users of Spike2 use the standard data capture and built-in analysis routines to capture data, and then export data, pictures and values using the clipboard or data files and make measurements using the cursors.

You can perform quite complex automated analyses without any scripting if you use Active Cursors. This feature allows you to search for features and plot and tabulate measurements. With judicious use of Channel Processes, Memory Buffers and virtual channels, complex data can be extracted without any use of scripting.

However, Spike2 can do a great deal more than is exposed by the menu-driven interface and often a few lines of script can automate a tedious manual procedure. If you suspect that a script could help your work, but you do not have the time or inclination to learn how to write one, you might consider using our script writing service. Contact CED for more information.

## How do I get started writing scripts?

Start by reading the Script manual as far as the alphabetical list of commands section. This should give you an overview of the script language. Next, look at the scripts supplied as part of Spike2. These will give you a flavour of how the language is used. You can look up any key word in a script by placing the text cursor in the keyword and then pressing the F1 key.

We supply a copy of the Spike2 Training Course manual as a PDF with each copy of Spike2. This contains many script examples and explanations. It also has tutorials covering a range of common scripting tasks.

There are also scripts available from our web site (Click [here](#), then click on Spike2 scripts in the left-hand panel). These range from tutorial examples to full applications.

We also run training courses in the UK and around the world. Contact CED for more information.

## Which version of Windows is best for Spike2?

Spike2 version 8 runs on Windows XP service pack 3 or later. It is available as a 64-bit program for 64-bit operating systems or a 32-bit build.

Spike2 version 7 runs under Windows XP service pack 2 or 3, Vista and Windows 7 and 8.

Spike2 version 6 will run on earlier versions of Windows but we would strongly advise you to upgrade to at least Windows XP.

## I get error -544 when I try to sample. Why?

The most usual cause for error -544 is that you are logged on as a User with insufficient privilege to increase your Working Set. It is unlikely to occur unless you are running Windows XP. This error code usually means that the system has been unable to lock down memory needed for data acquisition. Spike2 tries to avoid this problem by requesting an increase in the *Minimum Working Set* of the program, but this request can fail if:

- The size requested is too large for the system to operate safely.
- The user does not have the right to change the allocation (see below).

The *About Spike2...* command in the Help menu shows the current Minimum and Maximum Working Set sizes. Spike2 attempts to set the Minimum Working Set to 1000kB and the Maximum Working Set to 8000kB, but you can override these values in the registry. As this is an advanced option with system-wide implications, we do not provide an easy way to do this from inside Spike2.

### What is the Working Set Size

You can think of the Working Set Size as the amount of physical memory allocated to a process (in this case the Spike2 program). A program may use many megabytes of memory space (logical memory), but only the memory needed for immediate use has to be present in physical memory, the rest can be saved on disk and loaded when needed. In general, the more physical memory a program can use, the faster it runs as it does not need to wait for information to be loaded from disk.

However, Windows is a co-operative environment, and if any process (program) grabs a lot of physical memory, there is less for others to use, and in extreme cases, the system may grind to a halt. As far as the operating system is concerned, it likes all processes to use the minimum possible physical memory. Each process (like the Spike2 program), is allocated a minimum working set size and a maximum working set size which the operating system uses as a guide to how much physical memory a process uses.

These settings used to be more important when systems did not have much physical memory. Nowadays (2013), most systems have 4 GB of memory (if you have a 64-bit operating systems you can have much more), so you may want to greatly increase the (conservative) values we set by default.

### Minimum Working Set Size

This is the amount of physical memory that the operating system will always try to give a process, even when memory is running very low or your process is not running. If there are many processes running, the sum of the minimum working set sizes must always be less than the amount of physical memory in the system. The value given a process by the system by default depends on the amount of physical memory in the system and can be as low as 120 kB, and lower in some circumstances. Spike2 needs to lock down around 300 kB of memory when it samples data, plus needing memory for some code and data. If the minimum working set size is too small, the operating system will refuse to lock the memory for sampling, resulting in error -544.

### Maximum Working Set Size

The maximum working set size is the maximum amount of physical memory that the operating system normally lets your program have if memory is in short supply. If there is no other demand for memory, your process can have (much) more memory than this. The default values for this can be quite small, values of around 1 MB are common. If you run other applications at the same time as Spike2, you may be able to improve performance by setting a larger value.

### What Spike2 tries to do

When Spike2 starts up, it tries to set the minimum working set size to 1000 kB (enough to sample data and run), and the maximum size to 8 MB. You can override these sizes in the system registry. You must create and set the registry keys as DWORD values:

```
HKEY_CURRENT_USER\Software\CED\Spike2\Win32\Minimum working set
HKEY_CURRENT_USER\Software\CED\Spike2\Win32\Maximum working set
```

to the required sizes in kB that you need. If you do not supply a key, the default size will be used. If the size you request is too large for the system to grant, the request will be ignored for both parameters. Spike2 will never set the sizes less than the sizes the system gives you by default. If you need to find the default system sizes, set the sizes in the registry to 0, run Spike2 and open the About Spike2... window.

You can set these keys with RegEdit which is part of the Windows operating system. Spike2 does not create these keys, but will use their values if they exist. From version 4.08 of Spike2 you can set them with the following script:

```
Profile("Win32","Minimum working set", 1200); 'set new minimum size as 1200 kB
Profile("Win32","Maximum working set", 32000); 'set new maximum size to around 32 MB
```

Any change you make will not take effect until you exit Spike and restart it.

If you run large scripts that use a lot of memory, you can improve performance by increasing the maximum size, but if you overdo it, you will reduce system performance by starving other processes (including system processes) of memory.

### What to do if you cannot set the size

If the *Help* menu *About Spike2* box does not report 1000, 8000 kB (or the sizes you have requested in the registry), then you probably do not have sufficient “rights” to change the allocation. The following describes how we altered the rights on our system. Your system administrator will know how to do this, but they might find the following will save them some time checking the system manuals.

We describe how to do this for the local machine: it may be that your machine is administered remotely in which case the general description is correct, but the details may be different.

The new right you need is *Increase scheduling priority* (in low-level system documentation this is called the *SeIncreaseBasePriorityPrivilege* privilege and in programmer information it is also known as *SE\_INC\_BASE\_PRIORITY\_NAME*).

Rights are assigned to user Groups, so you must either add this right to the list of rights enjoyed by the Group you are a member of, or you must create a new Group with this right and become a member of it. In the descriptions below, steps to create a new group are labelled X. To extend the rights of an existing group, skip the steps labelled X and substitute the name of the existing Group for all mentions of *1401 Users*. The details vary with the operating system:

#### Windows XP

1. When logged on as Administrator, open the *Control panel* and then the *Administrative tools* folder.
  - X1. Run *Computer management*, select the entry *System tools:Local users and groups*.
  - X2. Select the *Groups* folder, use the *Action* menu to add a new local group called *1401 Users*.
  - X3. Use *Add...* to make the appropriate users into members of the new group.
  - X4. Exit from *Computer management*
2. Run *Local Security Policy*, select the entry *Local policies*, subentry *User rights assignment*.
3. Select *Increase scheduling priority*, use *Action : Security...* in NT 2000, use *Action : Properties* in XP.
4. Add *1401 Users* to the list of groups with that user right.
5. Exit from *Local Security Policy*.

#### Vista, 7, 8

A new right was added at Vista, "Increase a process working set", and this is normally set for all users, so from Vista onwards you should not need to make any changes for this. The instructions below are ONLY needed if your system administrator has (most unusually) decided that no users should have this right and has specifically disabled it for all users. If you follow these instructions you will only enable it for members of the *1401 Users* group.

1. When logged on as Administrator, open the *Control panel*, (switch to *Classic View* in Vista) and then open the *Administrative tools* folder.
  - X1. Run *Computer management*, select the entry *System tools:Local users and groups*.
  - X2. Select the *Groups* folder, use the *Action* menu to add a new local group called *1401 Users*.
  - X3. Use *Add...* to make the appropriate users into members of the new group.
  - X4. Exit from *Computer management*
2. Run *Local Security Policy*, select the entry *Local policies*, subentry *User rights assignment*.
3. Select *Increase a process working set*, use *Action : Properties*.
4. Click on *Object types...*, check the *Groups* box then click *OK*.

5. Add *1401 Users* to the list of groups with that user right.
6. Exit from *Local Security Policy*.

Next time you log on as a member of *1401 Users*, you should find that *Help>About Spike2* shows the expected memory available and that the error -544 no longer occurs.

### What to do if it still doesn't work

Don't panic! It works for everyone else, so there is a logical reason for failure. Please check that you have the latest service pack for your OS installed as out-of-date system software has been the cause of this in the past.

## Why can't I make contact with my CyberAmp from Spike2?

Assuming that the correct communications port is set in the Edit menu preferences, and that you installed Spike2 and selected CyberAmp support, the most common reason for lack of communication is that the rotary switch on the rear of the CyberAmp is not set to 0.

We have also seen problems on computers where the COM port has been set up in the computer BIOS as an infrared communication device.

## CyberAmp with input probes 401, 402, 405 and 414 has no output

These probes are described as differential in their specification, so you have selected the differential input setting. However these probes convert the signal to single ended. If you select the differential input, the same signal is applied to both CyberAmp differential inputs, and the result is to test the common mode rejection of the system. Select single ended or inverted inputs with these probes.

## Why can't I make Process in Gated mode work?

The Gated radio button in the on-line Process dialog has **nothing at all** to do with a stimulus channel used for PSTH analysis or event correlations or waveform averages. Many users who select the Gated radio button should have selected the Automatic button.

The purpose of the Process dialog is to select an area of data for analysis, defined by a start and end time. The fact that the analysis might use a stimulus channel, so only a small part of the data in the time range is used, is not a consideration for the Process dialog. If your analysis uses a stimulus channel (e.g. PSTH or Waveform average), then only the stimuli that lie in the time range set by Process are used.

Normally, you set a process region explicitly as a start and stop time. However, for on-line use we allow result views to update in three ways: Manually, Automatically, and Gated. In Gated mode, the data area to be processed is not defined in terms of a start and stop time. Instead, the data area is defined with respect to a Gate signal. Each time a gate is detected, the analysis set for the result view is performed on the data starting at the pre-gate time before the trigger for a time set by the Sweep length field. If your analysis also uses a stimulus channel, then the stimuli that lie in this time range are used.

The mistake that many users make is to assume that the gate signal is the same as the stimulus. At best this makes the analysis rather inefficient, at worst it can stop any analysis happening at all (for example if the pre-gate time is less than the pre-stimulus time set for the analysis).

## Sample rate limits and interface type

The maximum possible data transfer rate is determined by your 1401 type. However, this can be further limited by the way that your 1401 is connected to the host computer. There are four different possible connection methods, of which only three are supported for sampling in Spike2 version 8:

1. ISA interface. This is now obsolete and you are only likely to find it with a very old computer. If you have a Power1 or Micro2 with an ISA interface you will not be able to sample with it in Spike2. However, both of these devices can use USB, so you should use that instead. For interest only, the maximum transfer rate

from the 1401 to the host with this interface was around 250 kB per second.

2. PCI interface. This is increasingly rare, but your Power1 or Micro2 might be using it. Modern computers do not have PCI slots any more so you should use the USB interface instead. The maximum transfer rate with the PCI interface was around 1 MB per second.
3. USB1 interface. Some of the very early Power1 units had a USB 1 interface. This limits the maximum transfer rate to around 1 MB per second.
4. USB2 interface. All the supported 1401s (Power1-3 and Micro2 and Micro3) apart from a few very early Power1 devices have a USB 2 interface. The data transfer rate is much higher in the range 10 to 48 MB per second, depending on the 1401 type. However, if you plug your 1401 with USB 2 into a USB 1 port on your computer (older computers often have both types of USB port), your transfer rate will fall back to USB 1 speeds.

If your 1401 supports USB 2.0 and you have a USB 2.0 port available on your PC, then USB is the fastest interface and you should use that.

If your 1401 supports USB 1, you have a choice of interface as all 1401s with USB 1 can also be used with the CED PCI card. The PCI card is a little faster than the USB 1 interface, but this is unlikely to make much difference unless you are getting towards the limits of what can be achieved on your 1401.

## Ctrl+Shift+0 does not work

Microsoft have grabbed the `Ctrl+Shift+0` key combination for IME language control in Vista and Windows 7. You can get instructions to defeat this at this Microsoft link: <http://support.microsoft.com/kb/967893>

The Microsoft instructions are (in January 2012):

1. Click Start, and then click Control Panel.
2. Double-click Regional and Language Options.
3. Click Keyboards and Languages, and then click Change keyboards.
4. Click Advanced Key Settings, and select Between input languages.
5. Click change Key Sequence.
6. For Switch Keyboard Layout, select Not Assigned.
7. Click OK to close each dialog box.

We make use of this combination to scroll the x axis to locate a vertical cursor in a data view and to set all visible codes to 00 in the clustering dialog.

In Windows 8 the `Ctrl+Shift` combination is used by default to switch the keyboard layout. To disable this:

1. Go to Settings->Control panel->Language->Advanced settings.
2. Click Change Language bar hot keys.
3. In Text Services and Input Languages in the Advanced Key Settings tab select Between Input Languages, then click the Change Key Sequence button and disable the the key sequence. It should display as (None).

## I'm having trouble getting FilePathSet to work in Windows. Why?

The most common reason for his problem is that you have forgotten that the `\` character is treated specially by the script compiler when it is part of a literal string. Backslash is an escape character and is considered in combination with the following character. Not all character pairs generate acceptable "escape" sequences. You must use `\\` to produce a single `\` in the output. For example, to set the path to `c:\Spike2\data` you would need: `FilePathSet("c:\\Spike2\\data")`. A simple way to avoid this problem is to use the slash character `/` in place of the backslash, thus: `FilePathSet("c:/Spike2/data")`;

## On-line displays with my AGP card are very-very slow. Why?

This is of historic interest only; modern systems do not use this display type.

Some "economy" AGP cards use the system processor to scroll windows sideways. This is usually a quick operation. However, if you have 32-bit colour set (or even 16-bit colour with some cards), the time taken to scroll becomes significant, and the system becomes very unresponsive as it is spending all its time moving screen pixels. You can usually fix this by reducing the colour depth to 256 colours (right-click on the desktop, select Properties, then Settings and adjust the colour settings) and/or by reducing your screen resolution.

Alternatively, you could replace your AGP card with one that has a graphics processor so you can enjoy much faster scrolling with very little processor load at all colour depths.

## Spike2 marked as Not Responding

If a top level window (such as the Spike2 application) does not respond to Windows messages for about 5 seconds, it may be marked as Not Responding. This can happen when Spike2 is processing a lot of data, if a script is running without releasing any idle time, or if a drawing operation takes a very long time.

Since Windows XP, the operating system replaces the Not Responding window with a ghost window that copies the attributes of the real window and adds "Not Responding" to the title bar. This is done so that you can move the window and attempt to close it (if you think it is really dead). If the window subsequently comes back to life, the original window is restored and all is well.

### Never ending drawing

In Windows 7 but not in XP (we have not tested with Vista or Windows 8) a ghost window created in the middle of a screen update seems capable of causing the system to draw forever.

Unfortunately, it seems that if the application is in the middle of a Paint operation (updating an area of the application window that is no longer valid) when this occurs, the application is disconnected from the DC (Device Context - a logical representation of the screen that is being updated) it is painting to, and when the paint operation completes, the original DC does not think it has been painted. If the application then starts to respond to messages, Windows restores the original window, but immediately asks for another Paint operation for the same screen area, which puts us into a loop.

In Spike2 from version 7.03 onwards, if we detect that a time view drawing operation is taking more than a couple of seconds, we stop it early and mark the as yet unpainted area of the channel as invalid so that it gets painted in a subsequent Paint operation. However, this cannot be done if the channel is part of a group of overdrawn channels or if the window is in triggered overdraw mode or for a channel in **OverDraw WM** mode.

If your very slow screen update starts to loop approximately every 5 seconds, the only way out is to use the **Ctrl+Break** key combination to break out of the drawing operation.





# Index

- ' Comment designator 304
- # -
  - #include
    - in script 314
    - in sequence file 96
  - #IND 527
  - #INF 527
  - #QNaN 527
- \$ -
  - \$ string variable designator 299
- % -
  - % Integer variable designator 298
- & -
  - & in dialog prompts 399
  - & reference parameter designator 311
- \* -
  - \*= multiply and assign 304
- . -
  - .avi file extension 122
  - .cfxb file extension 122
  - .pls file extension 122
  - .s2c Configuration file extension
    - default.s2c 66
    - last.s2c 66
  - .s2cx Configuration file extension 122
    - Settings so double-click works 140
  - .s2rx Resource file extension 122
  - .s2s script file extension 122
  - .smr standard file extension 50, 122
  - .smrx standard file extension 50, 122
  - .srf Result file extension 122
  - .sxy XY file extension 122
  - .txt text file extension 122
- / -
  - /= divide and assign 304
- :-
  - := assignment 304
- ? -
  - ? in Ternary operator 23, 306
- [ -
  - [start:size] array syntax 300
- \ -
  - \ in path names (beware!) 770
  - \\ string literal escape character 299
- { -
  - { } optional syntax 291
- | -
  - | vertical bar 291
- + -
  - += add and assign 304
  - += Append string 304
- = -
  - = sequencer directive 93
- - -
  - = subtract and assign 304
- > -
  - > Display output sequencer text 90
- 1 -
  - 1401
    - Control 328, 596
    - Device driver revision 285
    - Monitor revision 285
    - Select unit 29
    - USB vs PCI interface 769
  - 1902
    - Connections 731
    - Get revision 385
    - Interactive support 728
    - script support 381
    - Serial line connection 166
- 3 -
  - 32-bit files 50
- - -
  - 544 error code 767
- 6 -
  - 64-bit files 50
  - 64-bit operating systems 30
- A -
  - Abort sampling 64, 543
  - About Spike2 767
  - Abs Max value
    - between cursors 266
  - ABS output instruction 111
  - Abs()
    - Script function 329
    - Virtual channel function 237, 238
  - Absolute value of expression or array 329
  - ACos calculation of 340
  - Active cursors 262
    - CursorActive() 389
    - CursorActiveGet() 390
    - CursorSearch() 393
    - CursorValid() 394
    - Position if search fails 262
    - Step right/left 25
  - Add cursor 392
  - Add data to end of file 373
  - Add items to memory buffer 229
  - ADD output instruction 112
  - Add to array 330
  - ADDAC output instruction 100
  - ADDACn output instruction 101
  - ADDI output instruction 111
  - Additional licences 4
  - AGP card problem 771
  - Aliasing of waveform data 36
  - All pass filter 712
  - All Poles 483
  - All stop filter 712
  - Alphabetical script index 329
  - Amplitude of power spectrum 332
  - Amplitude of waveform 375, 377
  - Analysis
    - Command synopsis 322
    - Create arbitrary result view 569
    - Event correlation 566
    - Identify target views/channels 602
    - Interval histogram 567
    - Number of items accumulated 583

- Analysis
  - Peri-stimulus time histogram (PSTH) 569
  - Phase histogram 568
  - Process all linked views 530
  - Process time view data 530
  - Shuffled event correlation 567
  - Waveform average and accumulate 566
  - Waveform correlation 570
  - Waveform correlation DC removal 570
- Analysis menu 206
  - Delete channel 245
  - Fit data 223
  - Measure to a channel 222
  - Measurements 217
  - Memory buffer 228
  - New Result View 206
  - Process settings 214
  - Save channel 245
- and 304
- AND output instruction 113
- ANDI output instruction 113
- ANGLE output instruction 104
- Annotate script view 204
- App() 329
- Append files 373
- Application close 432
- Application data path 31
- Application window handle 329
- Apply digital filter 710
- Arbitrary waveform output 57
  - Script function overview 326
  - Select waveform for graphical sequencer 83
  - Using the text sequencer 119
  - WAVEGO output instruction 119
  - WAVEST output instruction 120
- Arc Sine and Arc Cosine 340
- Arc tangent function 340
- Area 220, 266, 367
  - between cursors 266
  - Modulus between cursors 266
  - Modulus in trend plot 220
  - trend plot 220
  - under curve between cursors 266
  - under curve in trend plot 220
- Argument lists 311
- ArrAdd() 330
- Arrange icons 257
- Array arithmetic 323
  - array range designator 300
- Arrays 332
  - About 300
  - Absolute value 329
  - Add constant or array 330
  - Arc tangent function 340
  - Ceiling of real array elements 356
  - Command summary 323
  - Copy 330
  - Cosine of array 387
  - Cubic splines 337
  - Declaring 299
  - Difference of two arrays 338, 339
  - Differences between elements 331
  - Dimensions 300
  - Division 331
  - Dot product 332
  - Exponential function 420
  - FFT analysis 332
  - Fill with channel data 359
  - Fill with channel list 366
  - FIR filter 334
  - Floor of real array elements 461
  - Fractional part of real number 463
  - Gain and phase 332
  - Hyperbolic cosine of array 388
  - Hyperbolic sine 571
  - Hyperbolic tangent 587
  - Integrate 336
  - Inverse FFT 332
  - Length of array 483
  - Logarithm to base 10 488
  - Logarithm to base e 487
  - Matrix 300
  - Maximum value 497
  - Mean and standard deviation 339
  - Minimum value 507
  - Multiplication 336
  - Natural logarithm 487
  - Negate 336
  - Passing to functions 300
  - Power function 527
  - Power spectrum 332
  - Resample array 337
  - resize 302
  - Result view as an array 303
  - Set to constant 330
  - Sine of array elements 571
  - Smoothing and filtering 334
  - Sorting 336
  - Square root of array elements 574
  - Sub-array 300
  - Subtract array from value or array 338
  - Subtract value or array from array 339
  - Sum of product 332
  - Sum of values 339
  - Syntax 300
  - Tangent of the array elements 586
  - Total of array elements 339
  - Truncate real array elements 596
  - Vector 300
- ArrConst() 330
- ArrDiff() 331
- ArrDiv() 331
- ArrDivR() 331
- ArrDot() 332
- ArrFFT() 332
- ArrFilt() 334
- ArrHist() 335
- ArrIntgl() 336
- ArrMul() 336
- ArrSort() 336
- ArrSpline() 337
- ArrStats() 337
- ArrSub() 338
- ArrSubR() 339
- ArrSum() 339
- Asc() 340
- ASCII code of character 340
- ASCII code to string conversion 379
- ASin, calculation of 340
- Assignment 304
- ASz() sequencer expression 91
- ATan() 340
  - Virtual channel function 237, 238
- Audio codecs 751
- Audio recording 746
- Auto Format 156
- Auto units
  - X axis 173
  - Y axis 174
- Auto-complete 157
- Auto-correlation of events 208
- Auto-correlation of waveform 570
- Auto-correlation of waveforms 214
- Automatic file naming
  - Date and time specifiers 60
  - File size limit 60
  - File time limit 60
  - Prompt for a comment 60
  - Sequences of files 60

- Automatic file save at sample end 543, 544
  - Automatic file saving 60
  - Automatic file saving folder 429, 430
  - Automatic processing during sampling 531
  - Automatic scrolling 204, 612
  - Automatic update of result view 216
  - Auxiliary values 195
  - Auxiliary variables 195
  - Average of waveform 211
    - Number of sweeps 179
  - Average waveform data 566
  - avi file extension 122, 197
    - compress files offline 754
  - AVI file offset 198
  - avicom application 754
  - Axis as scale control 12
  - Axis as scroller 12
  - Axis controls 173
    - Clustering 698
    - Drawing colour 380
    - Palette colour 379
    - Show and Hide 176
    - XY view data tracking 615
  - Axis grid display control 466
- B -**
- Background bitmap 363
  - band 304
  - Band pass filter 708, 712
  - Band stop filter 712
  - beats per minute 189
  - Beep or tone output 572
  - Beeps 765
  - BEQ output instruction 110
  - Bessel filter 714
  - Beta function 341
  - BetaI() 341
  - Between cursors peak detect 652
  - BGE output instruction 110
  - BGT output instruction 110
  - Big file
    - Explanation 50
    - FileNew() 427
    - FileSaveAs() 432
    - Import 124
    - In Channels list 34
    - SampleBigFile() 545
  - Bin access in result view 303
  - Bin number to x axis unit conversion 351
  - Bin\_def.bib 124
  - Binary file commands 321
  - Binary files
    - Close 423
    - Little or big endian 352
    - Move current position 352
    - Open 428
    - Read data 351
    - Write data 354, 356
  - Binary import default format file 124
  - BinError() 349
  - Binomial coefficient 350
  - Binomial distribution 341
  - BinomialC() 350
  - Binsize() 350
  - BinToX() 351
  - Bitmap output
    - To clipboard 148, 416
    - To file 132, 432
  - bitwise operators 304
  - Black and White display 607
  - Black and white displays 199
  - BLE output instruction 110
  - BLT output instruction 110
  - BNE output instruction 110
  - BNZERO output instruction 99
  - Boltzmann sigmoid 444
  - Bookmarks 26, 155
    - Set on found text 154
  - bor 304
  - bpm display mode 189
  - BRAND output instruction 117
  - BRANDV output instruction 117
  - BRead() 351
  - BReadSize() 351
  - break 309
  - Break out of cursor calculations 266
  - Break points
    - Clear all 292
    - Set 292
  - Breaking out of a script 294
  - Breaking out of drawing 197
  - Breaking out of processing 215
  - Breaths per minute 189
  - BRWendian() 352
  - BSeek() 352
  - Buffer overflow 65
  - Bug fixes 759
  - Burst analysis
    - Burst statistics 354
    - Creating bursts from events 353
    - Revising bursts 353
  - Burst mode 46, 52
  - BurstMake() 353
  - BurstRevise() 353
  - BurstStats() 354
  - Butterworth filter 714
  - Buttons 588
  - BWrite() 354
  - BWriteSize() 356
  - bxor 304
  - Bxx output instructions 110
  - BZERO output instruction 99
- C -**
- C0-C9 cursor shortcut in dialogs 23
  - Calibrate waveform 357, 369, 375
    - Dialog 245
    - methods 246
  - CALL
    - CALLV and CALLn output instructions 108
  - Call stack in debug 296
  - Call tips 292, 310
    - Description and use of 161
    - Set display of 160
    - Style for 160, 162
    - Syntax for user-defined 160, 161
  - CANGLE output instruction 104
  - Caret
    - Get and set position 515
    - Get column number 515
    - Get line number 515
    - Get position and set relative 515
  - Caret colour 160
  - Cascade windows 256
  - case 307
  - Case sensitivity
    - In searches 154
    - Output sequencer 90
  - CED 1902 signal conditioner
    - Conditioner commands 381
    - Conditioner overview 728
  - CED Software help desk 284
  - CED Software Licence conditions 4
  - CED web page 758
  - CEDCOND.INI 731
  - CEDCOND.LOG file 166, 728
  - Ceil() 356
  - cfbx file extension 122

- Ch(n) virtual channel function 234
- Chain script 562
- CHAN output instruction 115
- Chan\$( ) 357
- ChanCalibrate( ) 357
- ChanColour( ) 358
- ChanColourGet( ) 358
- ChanColourSet( ) 359
- ChanComment\$( ) 359
- ChanData( ) 359
- ChanDelete( ) 360
- ChanDuplicate( ) 360
- ChanFit( ) 361
- ChanFitCoef( ) 362
- ChanFitShow( ) 363
- ChanFitValue( ) 363
- Changes to Spike2 759
- ChanHide( ) 363
- ChanImage( ) 363
- ChanIndex( ) 364
- ChanKey( ) 364
- ChanKind( ) 366
- ChanList( ) 366
- ChanMeasure( ) 367
- Channel 179
  - Add channel process 372
  - Add new XY view channel 620
  - Attach horizontal cursor 467
  - Background bitmap 363
  - Clear channel process 373
  - Comment 359
  - Comment (result view) 180
  - Copy channel process 373
  - Copy data from XY view 616
  - Copy into an array 359
  - Create 368
  - Delete 360
  - Duplicate 360
  - Event count 388
  - Find duplicate 416
  - Generate channel list 366
  - Get channel process information 373
  - Get colour 358
  - Get or set title 377
  - Get or set units 377
  - Get physical port 371
  - Get selected state 376
  - Get visible state 378
  - Groups 369
  - Hide (interactive) 176
  - Hide (script) 363
  - Information dialog (result view) 180
  - maximum and positions 508
  - Maximum time of item in channel 497
  - Measure region 367
  - Memory based channels 501
  - Modify channel process 372
  - Modify XY view channel settings 620
  - Number in data file 546
  - Order 369
  - Save to another channel or file 373
  - Selecting 376
  - Selecting in a dialog 403
  - Set colour 359
  - Set colour by palette 358
  - Show (interactive) 176
  - Show (script) 376
  - Time of next item 516
  - Time of previous item 482
  - Title (result view) 180
  - Type of a channel 366, 545
  - Units (result view) 180
  - Value at given position 377
  - Vertical space 378
  - Weight 378
  - Write wave data 379
- Channel arithmetic
  - Rectify 237
- Channel commands 317
- Channel display 199
  - Colour override 199
  - Cubic spline 194
  - Dots 188
  - Draw mode colour 199
  - Instantaneous frequency 189
  - Lines 188
  - Mean frequency 189
  - Order 163
  - Overdraw WaveMark mode 190
  - Raster mode 190
  - Rate 189
  - Sonogram mode 191
  - State mode 194
  - Text mode 195
  - Waveform mode 194
  - WaveMark mode 194
- Channel duplication 244
  - Comment (time view) 179
  - Information dialog (time view) 179
  - Title (time view) 179
  - Titles 179
- Units (time view) 179
- Channel lists 23
- Channel number
  - Channel modified 9
  - Display in red 188
  - Displayed in red 9, 247, 251
  - Drawing colour 380
  - Number show and hide 369
  - Palette colour 379
  - Selecting channels 9
  - Show and hide 369
- Channel parameters 35
- Channel process
  - Arbitrary waveform output 275
  - Calibration 246
  - Dialog 247
- Channel range 23
- Channel specifier 313, 565
- Channel specifiers 23
- Channel types 21
- Channels for sampling 34
- ChanNew( ) 368
- ChanNumbers( ) 369
- ChanOffset( ) 369
- ChanOrder( ) 369
- ChanPenWidth( ) 371
- ChanPixel( ) 371
- ChanPort( ) 371
- ChanProcessAdd( ) 372
- ChanProcessArg( ) 372
- ChanProcessClear( ) 373
- ChanProcessCopy( ) 373
- ChanProcessInfo( ) 373
- ChanSave( ) 373
- ChanScale( ) 375
- ChanSearch( ) 375
- ChanSelect( ) 376
- ChanShow( ) 376
- ChanTitle\$( ) 377
- ChanUnits\$( ) 377
- ChanValue( ) 377
- ChanVisible( ) 378
- ChanWeight( ) 378
- ChanWriteWave( ) 379
- Character code 379
- Character code (ASCII) 340
- Chebyshev type 1 filter 714
- Chebyshev type 2 filter 714
- Check box in a dialog 404
- Chi-squared probability function 464
- Chi-squared value 629

- Choose k from n 350
- Chr\$( ) 379
- Circular dots 163
- Clear template 645, 646
- Clear text or result view 153
- ClearType 198
- Clipboard
  - Copy and Cut data 416
  - Copy result view as text 149
  - Copy spike shapes 653
  - Copy Textmark data 274
  - Copy time view as text 150
  - Copy to 148
  - Copy XY view as text 149
  - Cut current selection to the clipboard 417
  - Cut text to 148
  - Get text 417
  - Paste 417
  - Paste image to channel background 363
  - Paste text 153
- Clock
  - front panel connection 71
- Clock output during sampling 71
- Clock rate for output sequencer 71
- Clock tick 20
- Close all associated windows 130, 256
- Close document 130
- Close Spike2 application 432
- Close window 423
- CLRC output instruction 106
- Clustering
  - Apply changes 690
  - Autoscale 694
  - Axis control 698
  - Class ellipsoids 683
  - Clustering dialog 683
  - Colour Fade with Z 695
  - Copy cluster image as bitmap 692
  - Copy cluster information as text 692
  - Copy data as text 693
  - Current event 683
  - Denisty plot settings 697
  - Display principal components 691
  - Dot size 695
  - Ellipse control 688
  - Ellipse radius 696
  - Equal Scales 694
  - Find Short Interval 696
  - from correlations 682
  - from errors 683
  - from measurements 679
  - Getting started 701
  - Index 676
  - Interval histogram 691
  - INTH settings 693
  - Introduction 676
  - Jiggle settings 698
  - K Means algorithm 703
  - K Means dialog 699
  - Mahalanobis distance 706
  - Match to classes 701
  - Menu commands 689
  - Minimum interval between events 696
  - Normal Mixtures algorithm 704
  - Normal Mixtures dialog 700
  - Online update 689
  - Online Update dialog 690
  - Principal Component Analysis 677
  - Rescale 694
  - Restore class codes 690
  - Select Principal Components 692
  - Set codes 699
  - Time range control 686
  - Tracking changes over time 686
  - Update online 689
  - User shapes 683
  - View along axis 697
  - View settings 693
  - while sampling 689
- Codec in multimedia recording 751
- Coefficients 334
- Coefficients of filters 717
- COFF output instruction 105
- Collision Analysis Mode 656, 658
  - Apply analysis 576
  - Get information 578
  - Important area 656
  - Important area (script) 577
- Colon
  - array range designator 300
- Colour dialog 199
  - Changes at version 7.07 201
  - Channel colours 199
  - Save colours in registry 168
  - Sonogram 202
- Colour fade
  - Clustering 695
  - Display trigger 183
- Colour palette
  - Change colours interactively 199
  - Get colour by script 517
  - Set by script 518
  - View index 199
- Colour( ) 379
- ColourGet( ) 380
- Colours of screen items 379, 380
  - Data channels 358, 359
  - Data channels palette colour 358
  - Force Black and White 607
  - Similar colours 163
  - Sonogram 202
  - View colours (script, get) 601
  - View colours (script, set) 601
  - XY view 614
- ColourSet( ) 380
- Column number in text view 515
- Com port
  - Conditioner conections 731
  - Script commands 328
  - SerialClose( ) 563
  - SerialCount( ) 563
  - SerialOpen( ) 563
  - SerialRead( ) 564
  - SerialWrite( ) 565
  - TextMark data 41
- Combinations 350
- Comma as decimal point 158
- Command line 534
- Command line options 29
- Comment
  - Data channel 35
  - File comment 179
  - File comment automatic prompting 543
  - File comment auto-prompt 60
  - Get and set channel comment 359
  - Get and set file comment 424
  - In script language 304
  - Output sequencer 90
- comment toggling 156
- Commit data file 543
- Common questions 764
- Compatibility of Spike2 versions
  - Data file formats 50
  - Settings for 168
- Compile output sequence 83
- Compile script 280, 292
- compress avi files 754
- CondFeature( ) 381
- CondFilter( ) 382
- CondFilterList( ) 382
- CondFilterType( ) 383
- CondGain( ) 383
- CondGainList( ) 384

- CondGet() 384
- Conditioner 166
  - CED1902 731
  - CEDCOND.INI settings file 731
  - CEDCOND.LOG file 166, 728
  - CyberAmp 731
  - Digitimer D360 731
  - Power1401 gain option 731
  - Sample menu 273
  - Serial line connection 166
  - Serial port 166, 728
- Conditioner commands 381
- CondOffset() 385
- CondOffsetLimit() 385
- CondRevision\$() 385
- CondSet() 386
- CondSourceList() 387
- CondType() 387
- Configuration files 122
  - Contents 66
  - Load and save 140
  - Load from script 428
  - Sample Bar 272
  - Sample Bar List dialog 272
  - Save from script 432
  - Sequence of operations to save 67
  - Suppress use of 427
- Connections 38
  - Conditioner COM ports 731
  - Digital i/o for sequencer 72
  - Event 0 and 1 165
  - Event inputs 38
  - Port for WaveMark data 560
  - Power1401 DACs 2 to 5 99
  - Sample and Play wave trigger 165
  - Sequencer clock output 71
  - serial 166
  - Serial port 728
  - Setting port for event channel 548
  - Setting port for waveform channel 559
  - Template match and DIGLOW 639
  - Waveform channels 36
  - Waveform output 99
- const 300
- Constant delarations
  - Output sequencer 93
  - script language 300
- Contacting CED 758
- continue 309
- Continuous sampling mode 54
- Convert
  - A number to a string 583
  - A string to a number 599
  - A string to upper case 598
  - Between channel types with MemImport() 503
  - Between script data types 299
  - Event to waveform 234, 419, 607
  - Foreign file format 124, 424
  - Number to a character 379
  - Parse string into variables 541
  - Parse string setup 541
  - RealMark to waveform 234
  - Result view bin to x axis units 351
  - Seconds to Spike2 time units 613
  - String to lower case 482
  - Waveform to WaveMark 653
  - X axis units to bins 613
- Copy
  - A script text string to the clipboard 416
  - Array or result view to another 330
  - Channel processing 373
  - Cluster data as bitmap 692
  - Cluster information as text 692
  - Clustering data as text 693
  - External file 425
  - Result view as text 149
  - Spike shape templates 653
  - Spreadsheet format 148
  - Text 148
  - The current selection to the clipboard 416
  - Time and XY views as pictures 148
  - Time view as text 150
  - Waveform to result view 566
  - XY view as text 149
- Copy channels 46, 52
- Correlation of events 208, 566
  - Number of sweeps 179
- Correlation of waveforms 214
- Cos()
  - Script function 387
  - Virtual channel function 237, 238
- Cosh() 388
- Cosine of expression 387
- Cosine wave output 102
- Count
  - Points in an XY channel 616
  - Points in XY circle 617
  - Points in XY rectangle 618
- Count events in time range 388
- Count of events 267
- Count() 388
- Covariance array 629
- CPHASE output instruction 104
- Crash on startup 534
- Crash recovery 734, 737
- Crash reporting 758
- CRATE output instruction 103
- CRATEW output instruction 104
- Create new channel 231
- Create new memory buffer 228
- Create new permanent channel 505
- Create new result view 565
- Create result view 569
- Create TextMark 274
- Creating new channels
  - By detecting spikes 653
  - In memory 501
  - On disk 368
- CRINC output instruction 106
- CRINCW output instruction 106
- Cross-correlation of events 566
- Cross-correlation of waveforms 570
- cSpc channel specifier 313, 565
- CSZ output instruction 102
- CSZINC output instruction 103
- Ctrl+Shift+0 does not work 770
- Cub()
  - Virtual channel function 238
- Cub() virtual channel function 237
- Cubic spline
  - Array of data 337
  - Copy Spreadsheet interpolation 148
  - Display mode (Result view) 195
  - Display mode (Time view) 194
  - Errors due to 247
  - Interpolate loaded template 660
  - Waveform average sweep alignment 211
- Cumulative Poisson probability function 464
- Curly brackets 291
- Current directory 429, 430
- Current view 289, 294, 600
- Cursor 0 260, 262
- Cursor 0 stepping 217
- Cursor commands 320
- Cursor labelling styles 261, 269
- Cursor menu 260
  - Active mode 263
  - Cursor regions 266
  - Delete 260
  - Delete Horizontal 268

- Cursor menu 260
    - Display all 261
    - Display all Horizontal 269
    - Fetch cursor 260
    - Move To cursor 260
    - Move To Horizontal 268
    - New cursor 260
    - New horizontal 268
    - Renumber cursors 262
    - Renumber horizontal cursors 269
    - Set cursor position 260
  - Cursor Regions 266
    - Open from script 392
  - Cursor Values 265
    - Open from script 392
  - Cursor() 389
    - In dialog 23
  - CursorActive() 389
  - CursorActiveGet() 390
  - CursorDelete() 390
  - CursorExists() 391
  - CursorLabel() 391
  - CursorLabelPos() 391
  - CursorNew() 392
  - CursorOpen() 392
  - CursorRenumber() 393
  - Cursors 262, 266, 468
    - Active 262
    - Adding a vertical cursor 260
    - Adding horizontal 268
    - Break out of long calculations 266
    - Channel for horizontal cursor 467
    - Copy value to Log view 267
    - Create new cursor 392
    - Ctrl+drag to drag label 10
    - Delete 390
    - Delete cursor 260, 268
    - Delete horizontal cursor 467
    - Display all 261, 269
    - Drawing colour 380
    - Get and set horizontal position 466
    - Horizontal cursor interactive commands 268
    - Horizontal cursor label position 468
    - Horizontal cursor label style 468
    - Label position 391
    - Label style 261, 269, 391
    - Move window to cursor 260
    - Move y axis to cursor 268
    - New horizontal cursor 468
    - Open cursor window from script 392
  - Palette colour 379
  - Position 389
  - Print cursor values 267
  - Renumber 262
  - Renumber cursors 393
  - Renumber Horizontal 269
  - Renumber horizontal cursors 469
  - Set cursor position 260
  - Set Label dialog 261
  - Static 263
  - User-defined style 391, 468
  - Valid and invalid 262
  - Value at 265
  - Values between 266
  - CursorSearch() 393
  - CursorSet() 393
  - CursorValid() 394
  - CursorVisible() 394
  - Curve fitting 223
    - Exponentials 444, 446
    - Gaussian 444, 448
    - Linear 450
    - Non-linear 452
    - Polynomial 444, 456
    - Sigmoid 444, 457
    - Sinusoid 444
    - Sinusoids 459
    - Testing the fit 226
  - Curve fitting from scripts 628
  - Cut text
    - Edit menu command 148
    - The current selection to the clipboard 417
  - CWAIT output instruction 105
  - CWCLR output instruction 106
  - CyberAmp
    - Connections 731
    - Get revision 385
    - Interactive support 728
    - problems 769
    - script support 381
- D -**
- D360 731
  - DAC connections for Power1401 99
  - DAC output during sampling 326
    - Arbitrary waveforms 57
    - Reset value before/after sampling 62
    - Reset value before/after sampling (script) 517
    - Sequencer DAC output 70
  - DAC output instruction 100
  - DAC output offline 275
  - DAC Silo 526
  - DACn output instruction 101
  - Damaged files
    - recovery (32-bit files) 737
    - recovery (64-bit files) 734
  - DANGLE output instruction 104
  - Data buffering 62
  - Data file
    - Read sampling configuration from 140
    - Size and time limit 50
    - Versions of Spike2 data files 20
  - Data points cursor mode 263
  - Data sampling 34
  - Data storage during sampling 765
  - Data types 297
    - Compatibility 299
  - Data view short cut keys 25
  - Date
    - as numbers 587
    - Change data file creation 437
    - data file creation 425, 436
    - In automatic file name 60
    - of item in a data file 425
  - Date\$() 394
  - dB scale 718
  - DBNZ output instruction 107
  - DBNZn output instruction 108
  - DC removal in waveform correlation 570
  - DC remove wave 247
  - Debounce 39
    - Digital Marker 40
    - Digital Marker (script) 547
    - Event channels 38
    - Event channels (script) 547
  - Debug
    - Call stack 296
    - Dump of internal objects 397
    - Enter debug on error 295
    - Enter debugger 294
    - Globals window 295
    - Heap information 396
    - Locals window 295
    - Operations 326
    - Options 398
    - Script commands 326
    - Set programmed break point 395
  - Debug bar 281
  - Debug script 293

- Debug() 395
- Debugging a script 292
- DebugHeap() 396
- DebugList() 397
- DebugOpts() 398
- Decibel scale 718
- Decimal point as comma 158
- Default arguments 311
- Default text style 160
- DEFAULT.S2C configuration 66, 199
- DEFAULT.S2CX configuration 140
- Delay
  - Fixed 81
  - Random equal density 81
  - Random Exponential 81
  - Random Poisson 81
- Delay in script 624
- DELAY output instruction 107
- Delete
  - Channel 360
  - Channels 206
  - Cursor 390
  - File 425
  - Horizontal cursor 467
  - Memory channel 502
  - Memory channel time range 502
  - Selection 416
  - Substring 399
  - XY view data 615
- Delete data channel 245
- Delete files from script 425
- Delete memory buffer items 230
- Delete selected text 153
- DelStr\$() 399
- Demonstration script 9
- Density plot settings (clustering) 697
- Determinant of a matrix 491
- DF() virtual channel function 235, 241
- diag() 300
- Diagonal of a matrix 300
- Dialog expressions 23
- Dialogs 399
  - & in dialog prompts 399
  - Buttons 402
  - Check box 404
  - Complex example 401
  - Create new dialog 405
  - Dialog units 399
  - Display and collect responses 409
  - Enable and disable items 405
  - Font 406
  - Get item value 412
  - Get position 406
  - Get time or x value 412
  - Group boxes 407
  - Idle function 402
  - Integer number input 407, 410
  - Real number input 409
  - Selecting a channel 403
  - Selecting one value from a list 408
  - Set label 408, 411
  - Show or hide items 412
  - Simple example 400
  - Simple format 399
  - Slider control 410
  - Text string input 411
  - Tooltips 399
  - User actions and call-backs 402
  - User-defined 399
- DIBEQ output instruction 98
  - Clash with template match 639
- DIBNE output instruction 98
- Dichotic notch 218
- Differences between array elements 331
- Differentiate wave 247
- Differentiating filter 708
- Differentiator filter 712
- DIGIN output instruction 99
- Digital filter 708
  - Dialog 709
  - FIR filter dialog 711
  - FIR filters 708
  - FIRMake filter types 719
  - IIR filter dialog 714
  - IIR filters 708
- Digital filter bank
  - Apply from script 437, 473
  - Create digital filter from script 439
  - Filter name from script 441, 478
  - Force filter calculation 438
  - Get filter bank information 440, 476
  - Sampling frequency range 441
  - Set attenuation of filter bank filter 438
  - Set comment in filter bank 439, 475
  - Set filter bank information 475
- Digital filtering 324
- Digital inputs 38
  - Connections 72
  - test bits 98
  - test saved bits 98
  - TTL levels 39
  - wait for bit pattern 99
- Digital marker 40
  - and sequencer connections 72
  - Connections 40
  - Link to output 41
- Digital outputs 72, 639
  - Connections 72
  - DIGLOW 98
  - DIGOUT 97
  - NDR and NDRL 72
  - Reset values before/after sampling 62
  - Reset values before/after sampling (script) 517
  - Sequencer update 71
  - Template match 639
  - WaveMark template match 639
- Digitimer D360 731
- DIGLOW output instruction 98
  - Template match clash 639
- DIGOUT output instruction 97
- Dimension of arrays 300
- Directories
  - For application data 31
  - For user data 31
- Directory for files
  - Get 429
  - Set (interactively) 430
- Directory for new data files 165
- DISBEQ output instruction 98
- DISBNE output instruction 98
- Disconnect Talker 273
- Display channel 376
- Display control 172
- Display overflow 186
- Display problem
  - Drawing repeats forever 771
  - Not responding 771
  - Slow scrolling 771
- Display trigger 183
- Distributions
  - Binomial 341
  - Chi-Squared 464
  - F distribution 346
  - Normal 464
  - Poisson 464
  - Student's 342
- DIV output instruction 112
  - Time penalty 71
- Division of arrays 331
- DlgAllow() 402
- DlgButton() 402
- DlgChan() 403



- DlgCheck() 404
  - DlgCreate() 405
  - DlgEnable() 405
  - DlgFont() 406
  - DlgGetPos() 406
  - DlgGroup() 407
  - DlgInteger() 407
  - DlgLabel() 408
  - DlgList() 408
  - DlgMouse() 409
  - DlgReal() 409
  - DlgShow() 409
  - DlgSlider() 410
  - DlgString() 411
  - DlgText() 411
  - DlgValue\$( ) 412
  - DlgValue() 412
  - DlgVisible() 412
  - DlgXValue() 412
  - docase 307
  - Dockable toolbars 608
    - Change floating size 609
    - Change state 610
    - Get docked state 609
    - Get handle 329
  - DOFF output instruction 105
  - Dominant frequency 235, 241
  - DOS command line 534
  - Dot product of arrays 332
  - Dot size
    - Clustering 695
    - Result view 195
    - Time view 188
    - XY view 196
  - Dots draw mode 188
  - Double-click 11
  - Down sample wave 247
  - Download latest Spike2 version 285
  - DPHASE output instruction 104
  - Dr Watson crash analyser 758
  - Drag and drop text 28
  - DRange() sequencer expression 91
  - DRATE output instruction 103
  - DRATEW output instruction 104
  - Draw a view 413
    - Break out of drawing 197
  - Draw() 413
  - DrawAll() 413
  - Drawing modes
    - Join XY view points 618
    - Result view 195
    - Time view 188
    - XY view 615
  - DrawMode() 413
  - DRINC output instruction 106
  - DRINCW output instruction 106
  - DrWtsn32.exe 758
  - DSZ output instruction 102
  - DSZINC output instruction 103
  - Dummy channels 46, 52
  - Dump internal script objects 397
  - Dup() 415
  - DupChan() 416
  - Duplicate
    - Sampling configuration channel 34
  - Duplicate channel 244
    - Based on template codes 658
    - Display all duplicates 176
    - Find duplicate from script 416
    - Script command 360
    - Titles of duplicate channels 179
  - Duplicate views 256
    - Get view handle 415
    - Interactive control 256
    - View duplicate from script 609
  - Duplicated event times 738
  - DWAIT output instruction 105
  - DWCLR output instruction 106
- E -**
- e
    - Logarithm to base e 487
    - Mathematical constant 306
    - To the power (exp) 420
  - Ec() virtual channel function 236
  - Edit marker codes 488
  - Edit memory channel 506
  - Edit menu 148
    - Auto Format 156
    - Clear 153
    - Copy 148
    - Copy as text 149
    - Copy Spreadsheet 148
    - Cut text 148
    - Delete selection 153
    - Find text 154
    - Paste 153
    - Preferences 158
    - Replace 155
    - Select All 153
    - Undo 148
  - Edit text
    - Bookmarks 155
    - copy and paste 27
    - Drag and drop 28
    - Find 155
    - Indent and outdent 27
    - Multiple selections 28
    - Rectangular select 29
    - Regular expressions 154
    - Search 154
    - Short cut keys 26
    - Text caret control 26
    - Virtual space 28
    - Wildcard searches 154
  - Edit TextMark dialog 274
  - Edit toolbar 155
  - Edit toolbar handle 329
  - Edit WaveMark data 655
  - EditClear() 416
  - EditCopy() 416
  - EditCut() 417
  - EditFind() 417
  - Editing commands 320
  - Editing script commands 320
  - EditPaste() 417
  - EditReplace() 417
  - EditSelectAll() 418
  - EEG processing 235
  - Eg() virtual channel function 236
  - Ellipse control in clustering 688
  - else
    - in case statement 307
    - in if statement 307
  - Email support 142
  - EMG processing 235
  - end 310
  - End of line characters
    - fixing 153
  - endcase 307
  - endif 307
  - Enlarge view 172
  - Environment variables 584
  - Environmental functions 327
  - Erase file 425
  - erf() 464
  - Error -544 767
  - Error bars
    - Script access 349
    - SEM 413
    - SetAverage() 566
    - SetResult() 569
  - Error codes as text 418

- Error function
    - erf(x) 464
  - Error\$( ) 418
  - Errors
    - Enter debug on error 295
    - Output sequencer compiler 86
    - Reporting crashes to CED 758
  - Es( ) virtual channel function 236
  - Esc key in a script 294
  - Escape character backslash 299
  - Escape from drawing 197
  - Escape key in a script
    - Enable/disable 395
  - Et( ) virtual channel function 236
  - Eval( ) 418
  - Evaluate 280
  - Evaluate a script line 280
  - Evaluate argument 418
  - Event
    - Convert to waveform 236
  - Event 0 and 1 connections
    - Digital input pins 38
    - Micro1401 and Power1401 165
  - Event 3 sampling trigger 39, 64
  - Event correlation 566
  - Event count 267
  - Event data
    - Burst formation 353
    - Channel types 21
    - Connections 38
    - Convert to waveform 419
    - Copy As Text and Export As format 151
    - Drawing colour 380
    - last 25, 173
    - Levels 38
    - Measurements and the drawing mode 168
    - Palette colour 379
    - Read times into an array 359
  - Events in time range 388
  - EventToWaveform( ) 419
  - Exchange data with another computer 142
  - Execute program 534
  - Exit 141
  - Exit from Spike2 432
  - Exit problem 765
  - Exp( ) 420
  - Exponential fit 361, 444, 446
  - Exponential function 420
    - As event kernel 244
  - Export data 132
    - As bitmap file 132
    - As clipboard text (spreadsheet) 148
    - As spike2 file 132
    - As spreadsheet text 132
    - As text file 132
    - As Windows Metafile 132, 163
    - TextMark 274
    - To clipboard 149
  - Export data file 432
  - ExportChanFormat( ) 421
  - ExportChanList( ) 421
  - ExportRectFormat( ) 422
  - ExportTextFormat( ) 422
  - Expressions 304
    - In dialogs 23
  - Extensions of file names 122
  - External exporter 132, 421, 432
  - External files 428
  - External program
    - Kill 534
    - Run 534
    - Status of 535
  - Extract fields from a string 541
  - Extreme value
    - trend plot 220
- F -**
- F distribution 346
  - Factorials 350, 488
  - Fade colour
    - Clustering 695
    - Display trigger 183
  - Feature search 375
    - Active cursors 389
    - Start cursor search 393
    - Test search 394
  - FFT (Fast Fourier Transform) 212
  - FFT analysis
    - Of arrays 332
    - Of waveform data 568
  - File 544
    - Automatic file commit 543
    - Automatic save at sample end 543, 544
    - Comment 424
    - Comment automatic prompting 543
    - Copy external file 425
    - Name for automatic filing 544
    - Name linked to view 427
    - Name of sampling configuration 547
  - New window 427
  - Open window or external file 428
  - Size of data file 435
  - File comments 179
    - Automatic prompt 60
  - File export to MATLAB 133
  - File format converters 124
  - File icons 2, 3
  - File menu 122
    - Close 130
    - Close and Link 130
    - Exit 141
    - Export 132
    - Global resources 129
    - Import 124
    - Load and save configurations 140
    - New File 122
    - Open 123
    - Page Setup 143
    - Print and Print Selection 145
    - Print screen 146
    - Resource files 130
    - Revert To Saved 131
    - Save and Save As 131
    - Send Mail 142
  - File name extensions
    - .avi file extension 122
    - .cfbx file extension 122
    - .pls file extension 122
    - .s2cx Configuration file extension 122
    - .s2rx Resource file extension 122
    - .s2s script file extension 122
    - .smr standard file extension 50, 122
    - .smrx standard file extension 50, 122
    - .srf Result file extension 122
    - .sxy XY file extension 122
    - .txt text file extension 122
  - File size
    - Big files 50
    - Data file 435
    - Limits 60
  - File system commands 321
  - FileApplyResource( )
    - Apply resource 423
  - FileClose( ) 423
  - FileComment\$( ) 424
  - FileConvert\$( ) 424
  - FileCopy( ) 425
  - FileDate\$( ) 425
  - FileDelete( ) 425

- FileGlobalResource() 426
  - FileList() 426
  - FileName\$( ) 427
  - FileNew() 427
  - FileOpen() 428
  - FilePath\$( ) 429
  - FilePathSet() 430
  - FilePrint() 430
  - FilePrintScreen() 431
  - FilePrintVisible() 431
  - FileQuit() 432
  - FileSave() 432
  - FileSaveAs() 432
  - FileSaveResource() 434
  - FileSize() 435
  - FileTime\$( ) 435
  - FileTimeBase() 435
  - FileTimeDate() 436
  - FileTimeDateSet() 437
  - Fill gaps in waveforms 247
  - FiltApply() 437
  - FiltAtten() 438
  - Filtbank.cfbx filter bank file 710
  - FiltCalc() 438
  - FiltComment\$( ) 439
  - FiltCreate() 439
  - Filter
    - Apply 710
  - Filter bank 710
  - Filter coefficients 334
  - Filter marker data 251, 489
  - Filter selection 708
  - Filtering 708
  - Filtering of arrays 334
  - FiltInfo() 440
  - FiltName\$( ) 441
  - FiltRange() 441
  - Find a view 601
  - Find feature 375
    - Active cursors 389
    - Start cursor search 393
    - Test search 394
  - Find func or proc 292
  - Find next/last keyboard command 25
  - Find Short Interval 696
  - Find text 148, 154, 417
  - Find zero crossing of a function 626
  - FIR filter 718
    - Apply from script 437
    - Coefficients 717
  - Dialog 709
  - Differentiator example 723
  - Filter bank 710
  - Frequencies 717
  - Frequency bands 718
  - Frequency response 443
  - Make coefficients 441
  - Make coefficients (simplified) 442
  - Maximum useful attenuation 718
  - Multiband example 722
  - Number of coefficients 720
  - Nyquist frequency 721
  - Overview 708
  - Ripple in bands 718
  - Script functions 324
  - Technical details 717
  - Transition region 718
  - Weighting 718
  - FIR filter of array 334
  - FIR filter types (script) 719
  - FIRMake() 441
  - FIRQuick() 442
  - FIRResponse() 443
  - Fit data 223
    - Coefficients 225
    - Results 225
    - R-square 226
    - Settings 224
    - Testing the fit 226
  - Fit Gabor function 455
  - FitCoef() 443
  - FitData() 444
  - FitExp() 446
  - FitGauss() 448
  - FitLine() 450
  - FitLinear() 450
  - FitNLUser() 452
    - More complicated example 455
    - Simple example 454
  - FitPoly() 456
  - FitSigmoid() 457
  - FitSin() 459
  - Fitting routines 631
  - FitValue() 461
  - Floating point number 297
  - Floor() 461
  - Flow of control statements 306
  - Flush sampled data to disk 60, 513, 560
  - FocusHandle() 462
  - Folder for files
    - Get 429
  - Set (interactively) 430
  - Folding
    - Script display settings 160
    - Sequencer text display settings 162
    - View menu command 203
  - Folding margin 203
  - Font selection
    - Get characteristics from script 462
    - Interactive 198
    - Quick size change 27
    - Set characteristics from script 462
    - Used in dialogs 198
    - User-defined dialogs 406
  - FontGet() 462
  - FontSet() 462
    - for 309
  - Forget Talker 273
  - Formatted text output 527, 529
  - Frac() 463
  - Fractional part of real number or array 463
  - FrontView() 463
  - F-test 346
  - func 310
  - Function argument lists 311
  - Functions and procedures 310
  - Functions as arguments 313
- G -**
- Gabor function 455
  - Gain 375
  - GammaP() 464
  - GammaQ() 464, 465
  - Gaps in waveforms 247
  - Gated update of result view 216
  - Gaussian
    - As event kernel 244
  - Gaussian fit 361, 444
  - Gaussian fitting 448
  - General information 20
  - Getting started 8
  - Getting started with scripts 288
  - Global resource files 129, 426
    - Update 130
  - Global variables 299
    - Debug 294
  - Globals window 295
  - Go to Proc or Func 292
  - Gotchas 764
  - Gradient of line 266
  - Graphical editing 76

- Graphical sequencer
    - Arbitrary waveforms 79
    - Branch on condition, probability or variable 79
    - Editor 73
    - Generate digital marker channel event 79
    - Interval or Hz for pulse train 79
    - Period or Hz in sinusoid 79
    - Pulse train 79
    - Pulse train with amplitude change 79
    - Ramp 79
    - Single pulse 79
    - Single pulse with amplitude change 79
    - Sinusoid 79
    - Variable arithmetic 79
    - Wait for time, condition or variable 79
    - Write as text sequence 73
    - Write as text sequence (script) 554
  - Grid
    - Set colour 199
    - Show and hide 176
  - Grid colour 380
    - Palette colour 379
  - Grid() 466
  - Group channels 174
  - Groups of channels 369
  - Gutter 203
    - Restore in ViewStandard() 605
  - Gutter() 466
- H -**
- H1-H9 horizontal cursor shortcut in dialogs 23
  - Halt 310
  - HALT output instruction 109
  - Hamming window 191
  - Handles in use 329
  - Hanning window 191
  - Hardware needed 2
  - HCursor() 466
    - In dialog 23
  - HCursorChan() 467
  - HCursorDelete() 467
  - HCursorExists() 467
  - HCursorLabel() 468
  - HCursorLabelPos() 468
  - HCursorNew() 468
  - HCursorRename() 469
  - Header and footer 143
    - Print screen 146
  - Heap information 396
  - Help 284
    - in a script 292
  - Help desk 284
  - Help menu 284
  - Help() 469
  - Hexadecimal marker code
    - Display control 491
    - Lookup table 41
  - Hexadecimal number format 298, 527
  - Hide a channel 363
  - Hide cursor 0 262
  - Hide window 256, 610
  - Hide X axis scroll bar 176, 613
  - Hide y axis 622
  - High pass filter 708, 712
  - High pass filter example 721
  - High-pass filter wave 247
  - Hilbert transformer 725
  - Histogram
    - DrawMode() 413
    - Rate drawing mode 189
    - Result view 195
    - XY draw mode 196
    - XYJoin() 618
  - Histogram from data array 335
  - hms 173
  - Hold triggered display 183
  - Horizontal cursor
    - Copy position to the clipboard 269
    - Create new cursor (script) 468
    - Delete cursor 467
    - Drag back into view 10
    - Get associated channel 467
    - Get cursor position (script) 466
    - Interactive commands 268
    - Link to spike sorting cursors 644
    - Renumber cursors 269
    - Renumber cursors (script) 469
    - Script language 320
    - Set label 269
    - Set label (script) 468
    - Set label position (script) 468
    - Set position 269
    - Shortcut key 25
    - Spike sorting 644
  - Horizontal cursor link to spike shapes 644
  - Horizontal titles and units 176
  - Host operating system 584
  - Hwr()
    - Virtual channel function 238
  - Hwr() virtual channel function 237
  - Hyperbolic cosine 388
  - Hyperbolic tangent 587
  - Hysteresis 263
  - Hz() sequencer expression 91
- I -**
- Icons
    - Arrange minimised windows 257
    - for files used in and by Spike2 2
  - Ideal waveform sampling rate 36
  - Idle function 167, 402, 588, 593, 624
  - if 307
  - If() virtual channel function 234
  - Ifc() virtual channel function 234
  - IIR filter 714
    - Apply from script 473
    - Apply to an array 472
    - Details 714
    - Dialog 709
    - Filter bank 710
    - Filter model 714
    - Filter order 714
    - Filter type 714
    - Get filter stability 471
    - Notch filter 714
    - Overview 469, 708
    - Read back information 471
    - Script commands 324
    - Stability 469
  - IIRApply() 473
  - IIRBp() 474
  - IIRBs() 474
  - IIRComment\$( ) 475
  - IIRCreate() 475
  - IIRHp() 476
  - IIRInfo() 476
  - IIRLp() 477
  - IIRName\$( ) 478
  - IIRNotch() 478
  - IIRReson() 478
  - Image behind channel 363
  - Immediate start 272
  - Import channel into memory buffer 230
  - Import data into memory channel 503
  - Import folder 124
  - Import foreign data file 124, 424
  - Impulse response 334, 717
  - Include files 314

- Include files 314
    - in script 314
    - in sequence file 96
  - Incomplete Beta function 341
  - IND 527
  - Indent text 293
  - Indents
    - script files 160
  - Index
    - Alphabetical script function index 329
    - Script language index 296
    - Script topics 316
  - INF 527
  - Infinity 527
  - Info on a result view 179
  - Inkey() 479
  - Input a single number 479
  - Input a string 480
  - Input from the keyboard 479
  - Input\$() 480
  - Input() 479
  - Insert data into memory channel 506
  - Installation 4
    - in C: directory 31
    - in Program Files 31
  - Instantaneous frequency 189, 413, 419
    - Drawing colour 380
    - Palette colour 379
  - InStr() 480
  - Instructions for output sequencer 89
  - Integer data type 298
  - Integer range 298
  - Integrate array 336
  - Interact() 481
  - Internationalisation (comma as decimal point) 158
  - Interpolate
    - Array of data 337
    - Channel data 232
  - Interpolate wave
    - Channel process 247
  - Interrupt drawing 197
  - Interrupting cursor window calculations 266
  - Interval histogram 206
    - Linked to clustering 691, 693
    - Number of intervals 179
  - Interval histogram (INTH) 567
  - INTH 206
  - Introduction to Spike2 2
  - Invalid cursors 262
  - Inverse FFT 332
  - Inverse of a function 626
  - Invert matrix 492
  - ISA card and sample speed 769
- J -**
- J3 clustering measure 703
  - Join data points in XY view 618
  - jump out of script loops 309
  - JUMP output instruction 109
  - Jump to event 173
- K -**
- K Means
    - Algorithm 703
    - Dialog 699
  - Kaiser window 191
  - Key for XY view 182
  - Key window 377, 605
    - Control of 364, 619
  - Keyboard control of sequencer 70, 90
  - Keyboard input 479
    - Interact() 481
    - ToolbarSet() 593
  - Keyboard markers
    - Channel 40
    - Restore channel 549
    - Special code FF 65
    - Special codes 54, 64
  - Keyboard shortcuts
    - Data views 25
    - Text view 26
  - Keyboard test 481
  - Keypress() 481
  - Keywords 297
  - Kind of channel 366, 545
  - Kurtosis 337
- L -**
- Label for cursor 261, 269
  - Label position of cursor 391
  - Label style of cursor 391, 468
  - Language index 296
  - LAST.S2C configuration 66
  - LAST.S2CX configuration 140
  - LastTime() 482
  - LCase\$() 482
  - LD1RAN output instruction 119
  - LDCNTn output instruction 108
  - Least-squares linear fit 361, 450
  - Left\$() 483
  - Legal characters in string input 480
  - Len() 483
  - Length of array or string 483
  - Level crossing 230
  - Level events
    - Channel types 21
    - Convert rising/falling edges to marker codes 01/00 503
    - Convert to virtual channel 234
    - Create from waveform 230
    - Drawing mode 188
    - Sampling 38
  - Licence 4
  - Limit text lines in Log 158
  - Limits
    - on array size due to available memory 302
    - on memory used by result views 303
    - on sampling time and file size 60
    - on script size 315
  - Line draw mode for events 188
  - Line numbers 204, 602
    - in text view 515
    - Style 160
  - Line style in XY views 196
  - Line thickness
    - Default 163
    - Printing 163
    - Script 371
    - Time and Result view 179
    - XY view 196
  - Linear fit 629
  - Linear fitting 450
  - Linear least-squares fit 361, 450
  - Linear Prediction 483
  - LinPred() 483
  - List of channels 366
  - List of files 426
  - List of views 603
  - literal string delimiter 299
  - Ln() 487
  - LnGamma() 488
  - Load and Run a script 280
    - on startup 29
  - Load templates 660
  - Local variables 299
    - Debug 294
  - Locals window 295
  - Lock a template 645
  - Lock y axes 174

- Log amplitude of the power spectrum in dB 332
  - Log view
    - Display debug options 398
    - Dump of internal objects 397
    - General information 765
    - Handle 488
    - Limit lines 158
    - Limit lines (from script) 604
    - Short cut keys 26
  - Log() 488
  - Logarithm to base 10 488
  - Logarithm to base e 487
  - Logarithmic axes
    - Number of decades 163
    - X axis 173
    - XAxisAttrib() 611
    - Y axis 174
    - YAxisAttrib() 622
  - Logarithmic scale 718
  - LogHandle() 488
  - Long drawing operations 197
  - Low pass differentiator filter 712
  - Low pass filter 708, 712
  - Low pass filter example 720
  - Lower case version of a string 482
- M -**
- Magnify a channel 8, 11
  - Magnify an area 11
  - Mahalanobis distance 706
  - Manual contents 17
  - Manual update of result view 216
  - MARK output instruction 41, 117
  - MarkEdit() 488
  - Marker codes 489
    - Colour for 199
    - ColourSet() 380
    - Edit codes 488
    - Hexadecimal only display 176, 188, 491
    - Set code to display 176, 188, 491
  - Marker data
    - Channel types 21
    - Colour for 199
    - ColourSet() 380
    - Connections 40
    - Copy As Text and Export As format 151
    - Digital marker channel 40
    - Filter markers 251
    - Hexadecimal only display 491
    - Set code to display 188, 491
    - Set codes 252, 491
    - State display mode 194
  - Marker filter 206, 251
    - in Spike shape dialogs 652
  - MarkInfo() 489
  - MarkMask() 489
  - MarkSet() 491
  - MarkShow() 491
  - Mask for marker data 489
  - Match channel 247
  - MATDet() 491
  - Mathematical constants 306
  - Mathematical functions 323
  - MATInv() 492
  - MATLAB file export 133
    - Event data 137
    - File format 136
    - Marker data 137
    - RealMark data 138
    - Result data 139
    - Result view data 136
    - Script support 139
    - TextMark data 137
    - Time view data 134
    - Waveform and RealWave 136
    - WaveMark data 138
    - Workspace variable naming 133
    - XY data 139
    - XY view data 136
  - MATLAB script support 492
  - MatLabClose() 494
  - MatLabEval() 496
  - MatLabGet() 495
  - MatLabOpen() 494
  - MatLabPut() 494
  - MatLabShow() 496
  - MATMul() 496
  - Matrix 300
    - Determinant of 491
    - Diagonal of 300
    - Inverse of 492
    - Multiplication 496
    - Solve linear equations 496
    - Transpose of 300, 497
  - Matrix arithmetic 323
  - MATSolve() 496
  - MATTrace() 497
  - MATTrans() 497
  - Max()
    - Script function 497
    - virtual channel function 237
  - Maximum and Minimum cursor modes 263
  - Maximum and minimum of XY channel 620
  - Maximum Entropy 483
  - Maximum event rate 62
  - Maximum of channel or result view 508
  - Maximum possible run time 46
  - Maximum sustained event rate 38, 40, 43
  - Maximum total sampling rate 36
  - Maximum value 497
    - between cursors 266
    - trend plot 220
  - Maximum Working Set Size 767
  - Maxtime() 497
  - Mean
    - ChanMeasure() 367
    - Curve fitting 628
  - Mean event rate 267
  - Mean frequency 189, 413, 419
    - Drawing colour 380
    - Palette colour 379
  - Mean frequency (power spectrum) 241
  - Mean frequency (spectral) 240
  - Mean value
    - Between cursors 266
    - trend plot 220
  - Measure to a channel 222
  - Measure to XY dialog 217
  - Measure with mouse 11
  - MeasureChan() 498
  - Measurements
    - Cursor positions 269
    - MeasureChan() 498
    - MeasureToXY() 499
    - MeasureX() 500
    - MeasureY() 501
    - Tabulated output of 149
    - To a data channel 222
    - To an XY view 217
    - Types 220
    - With mouse on screen 11
  - MeasureToChan() 498
  - MeasureToXY() 499
  - MeasureX() 500
  - MeasureY() 501
  - Median filter 247
  - Median frequency 240
  - MemChan() 501
  - MemDeleteItem() 502

- MemDeleteTime() 502
  - MemGetItem() 503
  - MemImport() 503
  - Memory buffer 206, 228
    - Add items 229
    - Create new channel 228
    - Delete items 230
    - Enable/disable warning on file close 159
    - Warn on file close 232
    - Write to file 231
  - Memory channels
    - Add or edit data 506
    - Creating 501
    - Delete 502
    - Delete time range 502
    - Get item data 503
    - Import data 503
    - Write to file 505
  - MemSave() 505
  - MemSetItem() 506
  - Menus
    - Help 284
  - Message() 507
  - Metafile image export 653
  - Metafile output
    - To clipboard 148, 416
    - To file 132
  - Metafile output resolution 163
  - MF() virtual channel function 235, 240
  - Microseconds per time unit 46, 52
    - Change 435
    - Setting for new file 427
  - Mid\$() 507
  - Min() 507
    - virtual channel function 237
  - Minimum interval between clustered events 696
  - Minimum of channel or result view 508
  - Minimum value 507
    - between cursors 266
    - trend plot 220
  - Minimum Working Set Size 767
  - Minimum/Maximum 367
  - Minmax() 508
  - MMAudio() 509
  - MMFrame() 509
  - MMImage() 510
  - MMOffset() 510
  - MMOpen() 511
  - MMPosition() 511
  - MMRate() 512
  - MMVideo() 512
  - Modified() 513
  - Modulus (measure) 367
  - Monitor spike shapes 662
  - Monitor templates online 659
  - Monochrome display 607
  - Mouse
    - Channel overdraw 14
    - Channel spacing 14
    - Create pointer 514
    - Find position 590
    - Initial position in dialog 409
    - Measurements 11
    - Mouse wheel 173
    - Scroll X Axis 173
    - Select in OverDraw WM mode 190
    - ToolbarMouse() 590
    - unresponsive on-line 771
  - MousePointer() 514
  - MOV output instruction 111
  - Move channels 369
  - Move in text and cursor windows 515
  - MoveBy() 515
  - MoveTo() 515
  - MOVI output instruction 111
  - MOVRND output instruction 118
  - ms as time scaler 23
  - ms() sequencer expression 91
  - msTick() sequencer expression 91
  - MUL and MULI output instructions 112
  - Multimedia files 197
    - compress offline 754
    - Offset 198
  - Multimedia recording 746
    - Compress data 749
    - Configuration dialog 749
    - Disable drift compensation 753
    - Getting started 747
    - Recording data 754
    - Set slow frame rate 749
    - System requirements 747
    - Use slow frame rate 749
    - Video Capture dialog 748
  - Multimedia sound output 572
    - Speech 573
  - Multimedia window
    - Audio information 509
    - Frame positions 509
    - Frame rate 512
    - Frame stepping 511
    - Open window 511
  - RGB data access 510
    - Set and get play position 511
    - Set and get play state 511
    - Time offset 510
    - Video information 512
  - Multiple 1401s 29
  - Multiple monitor support 328, 406, 584, 610
    - Set mouse pointer position in dialog 409
  - Multiple selections of text 28
  - Multiple software licences 4
  - Multiplication
    - arrays 336
    - Matrices 496
- N -**
- Name format 297
  - Name of file linked to view 427
  - Natural logarithm 487
  - NDR and NDRL digital output signals 72
  - NEG output instruction 111
  - Negate array 336
  - New cursor 392
  - New document 63, 122
    - Temporary directory for data files 165
  - New file 427
  - New file from existing file 132
  - New horizontal cursor 468
  - New result view 206, 565
    - next 309
  - Next data point search 263
  - NextTime() 516
  - Non-linear fit 629
  - Non-linear fitting 452
  - NOP output instruction 110
  - Normal distribution 628
    - Probabilities 464
  - Normal Mixtures
    - Dialog 700
  - Normal settings for a view 605
  - Not a Number 527
  - Not Responding 771
  - Not saving to disk colour 199
  - Notch filter 714
  - Novell server 767
  - Number format (comma for decimal point) 158
  - Numeric expressions in dialogs 23
  - Numeric input 479

**- O -**

Off-line templates 659, 660, 661  
 OFFSET output instruction 105  
 Offset waveform 369  
 One and a half high pass filter 712  
 One and a half low pass filter 712  
 Online clustering  
   Delete all spikes 692  
   Update dialog 690  
 On-line template setup dialog 642  
 On-line templates 659, 660, 661  
 Open file 428  
 Open file problem 765  
 Opening  
   Files from command line 29  
   New document 63, 122  
   Old document 123  
 Operating system 584  
 Operators 304  
   bitwise 304  
   diag() diagonal of matrix 300  
   In dialogs 23  
   Matrix 300  
   numeric 304  
   Precedence (numeric) 304  
   Precedence (string) 305  
   string 305  
   ternary 306  
   trans() transpose matrix 300  
 Optimise the display 516  
 Optimise Y axis 174  
 Optimise() 516  
 Options for XY view 182  
 or 304  
 OR output instruction 113  
 Order of channels 163, 369  
 ORI output instruction 113  
 Oscilloscope style trigger 183  
 Oscilloscope style triggered display 606  
 Out  
   front panel connection 71  
 Outdent text 293  
 Output sequencer 70, 73, 76  
   64-bit times 554  
   Access a sampled channel 115  
   Access to data capture 115  
   Add constant to variable 111  
   Add digital marker as if sampled 117  
   Arbitrary waveform output contro 119

Bitwise AND 113  
 Bitwise OR 113  
 Bitwise XOR 113  
 Calculate variable values 92, 97, 98, 99, 101, 106  
 Change sequence during sampling 83  
 Change sequence during sampling (script) 554  
 Change tick period (script) 552  
 Changing the sequence during sampling 96  
 Clock rate 71, 73  
 Compare variable 110  
 Compatibility with older versions 88  
 Compile sequence 83  
 Compiler errors 86  
 Constants 93  
 Control panel 70  
 Copy variable 111  
 DAC outputs 99  
 DAC scaling 73, 93  
 Disable interactive jumps 70, 73  
 Display settings 162  
 Display text 90  
 Divide variable 112  
 Expressions 91  
 Format text 83  
 Format with step numbers 83  
 Get current sample time 116  
 Get current step (script) 553  
 Get file name (script) 555  
 Get sequencer mode (script) 554  
 Getting started 85  
 Graphical editing 76  
 Graphical editor setup 73  
 Graphical palette 78  
 Index 70  
 Instruction format 90  
 Instructions 89  
 Keyboard link control (script) 553  
 milliseconds per step 71, 73, 93  
 Minimum instruction and Table space 56  
 Minimum sequence size (script) 553  
 Minimum table size (script) 553  
 Multiply variables 112  
 Negate variable 111  
 Randomisation 117  
 Read only files 85  
 Reciprocal of variable 112  
 SCLK, SDAC, SET directives 93

Script control 552, 553, 554, 555  
 Set and set variable (script) 555  
 Set file name (script) 554  
 Set file to use 56, 83  
 Set reference time for TICKS 116  
 Set sequencer mode (script) 554  
 Set variable value 111  
 Special variables 92  
 State machine 109  
 TABDAT directive 94  
 Table access (script) 553  
 Table of values 94  
 TABSZ directive 94  
 Text sequence example 85  
 Timing faults 76  
 Variable arithmetic 110  
 Variable logic 113  
 Variable sum and difference 112  
 Variables 92  
 OutputReset() 517  
 Overdraw data 632  
   3D drawing setup 605  
   Add trigger times 604  
   based on channels 14  
   based on view triggers 183, 185  
   based on view triggers (script) 606  
   Clear list 187  
   Enable 3D drawing 606  
   Lock channel axes 174  
   Overdraw 3D dialog 186  
   Overdraw WaveMark mode 190  
   Time view overdraw list details 187  
   Using an XY view 632  
 Overdraw WaveMark display mode 190  
 Overwrite wave data 379

**- P -**

Page Setup dialog 143  
 Paged display on-line 183  
 Palette for colour 518  
 PaletteGet() 517  
 PaletteSet() 518  
 Parent and child directories/folders 426  
 Pass band 718  
 Passing arguments 311  
   by reference 311  
   by value 311  
   default values 311  
   functions and procedures 313  
 Paste from clipboard 417  
 Paste text 153  
 Path for file operations 429, 430



- Path problems in Windows 770
  - Paths
    - For application data 31
    - For user data 31
  - PCA 676
  - PCA() 518
  - PCI vs USB interface 769
  - Peak and trough
    - between cursors 266
    - trend plot 220
  - Peak search 230
    - cursor mode 263
  - Peak to peak
    - between cursors 266
    - trend plot 220
  - Peak to Peak value 367
  - Pen Width
    - Script 371
  - Pen Width dialog 179
  - per minute rates 189
  - Peri-stimulus time histogram 207
    - Number of sweeps 179
  - Peri-stimulus time histogram (PSTH) 569
  - Permutations 350
  - Phase display (FFT) in result view 332
  - Phase histogram 210, 568
    - Number of cycles 179
  - Phase of power spectrum 332
  - PHASE output instruction 104
  - pi
    - Mathematical constant 306, 586
    - Radians 387, 571
  - Pixels 371
  - Play wave bar handle 329
  - Play wave during output 326
  - Play waveform 59, 275
  - Play waveform toolbar 57
  - PlayOffline() 519
  - PlayWaveAdd() 520
  - PlayWaveChans() 522
  - PlayWaveCopy() 522
  - PlayWaveCycles() 523
  - PlayWaveDelete() 523
  - PlayWaveEnable() 523
  - PlayWaveInfo\$() 523
  - PlayWaveKey2\$() 524
  - PlayWaveLabel\$() 524
  - PlayWaveLink\$() 524
  - PlayWavePoints() 525
  - PlayWaveRate() 525
  - PlayWaveSpeed() 526
  - PlayWaveStatus\$() 526
  - PlayWaveStop() 526
  - PlayWaveTrigger() 527
  - pls file extension 122
  - Point style in XY views 196
  - Poisson distribution 464
  - Poly()
    - Virtual channel function 238, 239
  - Poly() virtual channel function 237
  - Polynomial fit 361, 444
  - Polynomial fitting 456
  - Pop-up tips in script views 161
  - Port
    - For WaveMark data 560
    - Setting for event channel 548
    - Setting for waveform channel 559
  - Port for sampling 35
  - Position of cursor 389
  - Position of window 608, 609
  - Pow() 527
  - Power function 527
  - Power in a band 235
  - Power in band 241
  - Power spectra
    - Of arrays 332
    - Of waveform channels 568
  - Power spectrum 212
  - Power1401
    - Programmable gain option 728
    - Programmable gain option 731
  - Preferences
    - Dialog 158
    - Edit menu 158
    - Saved in configuration files 66
    - Script access 531
  - Principal Component Analysis 518, 676, 677
    - Mathematics 679
  - Print
    - All views on screen 431
    - Formatted text output 527, 529
    - Print screen 146
    - Print visible region 431
    - Range of data 430
    - To log window 529
  - Print control 122
  - Print\$() 529
  - Print() 527
  - Printing 143
    - File name 143
  - From a script 529
  - Header and footer 143
  - Line thickness 163
  - Preview printed output 145
  - Spike shape templates 653
  - Time and date 143
  - To string 529
  - PrintLog() 529
  - Problems 764
  - proc 310
  - Procedures as arguments 313
  - Process dialog 215
    - for New file 64, 216
  - Process settings 214
  - Process time view data
    - Identify target views/channels 602
    - Process() script command 530
  - Process triggered 769
  - Process() 530
  - ProcessAll() 530
  - ProcessAuto() 531
  - ProcessTriggered() 531
  - Profile() 531
  - ProgKill() 534
  - Program Files installation 31
  - Programmable signal conditioners 728
  - ProgRun() 534
  - ProgStatus() 535
  - Prompt to save result and XY views 158
  - PSTH 207
  - Pw() virtual channel function 235
    - Interactive dialog 239
- Q -**
- QNaN 527
  - Query() 535
  - Questions 764
  - Quick channels 46, 52
  - Quiet startup 29
  - Quit Spike2 432
- R -**
- Radians
    - ATan() 340
    - Convert to degrees 586
    - Cos() 387
    - Sin() 571
    - Tan() 586
  - RAMP output instruction 101
  - Rand() 536

- RandExp() 536
- RandNorm() 537
- Random number generator 536
  - Exponential distribution 536
  - Normal distribution 537
- Randomisation in output sequencer 117
- Range of data points in XY view 620
- Range of integers 298
- Raster display
  - Drawing modes 195
  - Event correlation 208
  - PSTH 207
- Raster drawing mode 413
  - Drawing colour 380
  - in a result view 566, 568, 569
  - Palette colour 379
- Raster event display mode 190
- Raster sweep values 207, 208, 210
- RasterAux() 537
- RasterGet() 538
- RasterSet() 538
- RasterSort() 539
- RasterSymbol() 539
- Rate drawing mode 189
  - Drawing colour 380
  - Palette colour 379
  - Set from script 413
  - Setting interactively 188
- RATE output instruction 103
- RATEW output instruction 104
- Read binary data 351
- Read channel into an array 359
- Read only 123, 513
- Read text file
  - Input from a text file into variable(s) 539
  - Open file 428
- Read() 539
- Read-only files 123
- ReadSetup() 541
- ReadStr() 541
- Real data type 297
- RealMark data
  - Channel types 21
  - Convert to waveform 234
  - Copy As Text format 151
  - Export As format 151
  - Get and set display index 364
  - Get information 489
  - Sampling 42
- RealWave data 43
- Channel types 21
- Export as Adc data 132
- Rescale 179
  - use as integer 36
  - Write to channel 379
- RECIP output instruction 112
  - Time penalty 71
- Reciprocal of array 331
- Reclassify WaveMark data 655
- Reclassify WaveMarks dialog 656
- Record a script 280
- Record actions 280
- Record multimedia 746
- Recover data files 734, 737
- Rectangular selection of text 29
- Rectify waveform
  - Channel process 247
- Reduce View 172
- Reference parameters 311
- Registry
  - Conditioner settings 731
- Registry access 531
- Regular expressions 154
- Relative measurements 265
- Renumber cursors 262, 393
- Renumber horizontal cursors 269, 469
- repeat 308
- Replace matched text 155
- Replace text 417
- Replay waveforms 275
- Repolarisation cursor mode 263
- REPORT output instruction 41, 117
- ReRun a file 541
- Rerun an existing file 204
  - linked to play waveform 275
- ReRun() 541
- Resample wave 247
- Reset
  - DAC outputs 62
  - DAC outputs (script) 517
  - Digital outputs 62
  - Digital outputs (script) 517
- Reset sampling 64
- Residuals 629
- resize 302
- Resolution in time 46
- Resonator filter 714
- Resource files 122
  - Apply and save 130
  - Save views linked to time views 130, 257
- Resource information suppression 428
- Result files (.srf) 122
- Result view 195, 303
  - Access to contents 303
  - Array access 300, 303
  - Background palette colour 379
  - Bin width 350
  - Clear 153
  - Convert x axis units to bin number 613
  - Copy as text 149
  - Create user-defined view 569
  - Creating a new view 206
  - Draw modes 195
  - Drawing colours 380
  - Error bars 349
  - Get or set units 377
  - Getting started 16
    - maximum and positions 508
  - Modified 513
  - Number of items accumulated 583
  - Number of sweeps or items 179
  - Open file from script 428
  - Printing 430
  - Process data 530
  - Prompt to save unsaved view 158
  - Rasters 195
    - Script commands 318
    - sum and slope 266
    - Sum of bins 388
    - Update mode 216
      - Value at given x axis position 377
  - return 310
- RETURN output instruction 108
- Revert text document to last saved 131
- Revision 8.00 762
- Revision 8.01 760
- Revision history 759
- RGB colours 201
- Right\$() 542
- Rightmost characters from a string 542
- RINC output instruction 106
- RINCW output instruction 106
- Rm() virtual channel function 234
- Rmc() virtual channel function 234
- RMS Amplitude 367
  - between cursors 266
  - Calibration method 246, 357
  - Channel process option 247, 372
  - Trend plot 220, 500
- ROM revisions in 1401 741
- Root of equation 626

- Rotating the cluster window 688
  - Round a real to nearest whole number 542
  - Round dots 163
  - Round() 542
  - rpm 189
  - RS232 script commands 328
    - TextMark data 557
  - R-square 226, 361
  - Run external program 534
  - Run script 280, 292
    - after current script ends 562
    - command line 29
    - from Script Bar 281
    - short-cut key 281
    - startup.s2s 29
- S -**
- s() sequencer expression 91
  - s2cx Configuration file extension 122
  - s2rx Resource file extension 122
  - s2s script file extension 122
  - s2video application 197, 746
    - Detect if running 512
  - S64Fix data recovery utility 734
  - Sample Bar
    - Immediate start 272
    - Label and comment 60, 61, 547
    - Write to disk 272
  - Sample bar handle 329
  - Sample Bar List dialog 272
  - Sample control panel handle 329
  - Sample control toolbar 64
  - Sample data
    - how to 765
  - Sample interval 21, 36
  - Sample menu 272
    - Create a TextMark 274
    - Sample Bar 272
    - Sample Bar List dialog 272
    - Sampling configuration 272
    - Sequencer controls 274
    - Signal conditioner setup 273
    - Waveform output 275
  - Sample rate for waveform data 21, 36
  - Sample Status bar 66
    - Script access 329, 548
  - Sample toolbar
    - Control from script language 544
  - SampleAbort() 543
  - SampleAutoComment() 543
  - SampleAutoCommit() 543
  - SampleAutoFile() 543
  - SampleAutoName\$( ) 544
  - SampleBar() 544
  - SampleBigFile() 545
  - SampleCalibrate() 545
  - SampleChanInfo() 545
  - SampleChannels() 546
  - SampleClear() 546
  - SampleComment\$( ) 547
  - SampleConfig\$( ) 547
  - SampleDebounce() 547
  - SampleDigMark() 548
  - SampleEvent() 548
  - SampleHandle() 548
  - SampleIdle() 549
  - SampleKey() 549
  - SampleKeyMark() 549
  - SampleLimitSize() 549
  - SampleLimitTime() 550
  - SampleMode() 550
  - SampleOptimise() 550
  - SampleRepeats() 552
  - SampleReset() 552
  - SampleSeqClock() 552
  - SampleSeqCtrl() 553
  - SampleSeqStep() 553
  - SampleSeqTable() 553
  - SampleSeqTick0() 554
  - SampleSequencer\$( ) 555
  - SampleSequencer() 554
  - SampleSeqVar() 555
  - SampleStart() 555
  - SampleStartTrigger() 556
  - SampleStatus() 556
  - SampleStop() 556
  - SampleTalker() 556
  - SampleText() 557
  - SampleTextMark() 557
  - SampleTimePerAdc() 558
  - SampleTitle\$( ) 558
  - SampleTrigger() 558
  - SampleUsPerTime() 559
  - SampleWaveform() 559
  - SampleWaveMark() 560
  - SampleWrite() 560
  - Sampling 64
    - 1401 usage 66
    - Automatic processing 531
    - Burst mode 46, 52
    - Change output sequence 275
    - Cluster updates during 689
    - Controls during 64, 273
    - CPU usage 66
    - Data transfer rate 66
    - Diagnosing problems 741
    - Digital outputs 70
    - Disk space remaining 66
    - Error -544 when I try to sample 767
    - File size 435
    - File size limit 60
    - Get sample start time and date 436
    - Maximum rate 36
    - Multimedia data 754
    - Multiple files 60
    - Naming data file from a script 427, 432
    - Output during 70
    - Record script equivalent 34
    - Runtime control functions 325
    - Set sample start time and date 437
    - Setting where data is stored during sampling 430
    - Status bar 66
    - Threads 167
    - Time limit 60
    - Time remaining 66
    - Topics 34
    - Triggered start 64, 555, 556
    - View handles 548
  - Sampling configuration 34, 35, 66, 272
    - Add a new channel 35
    - Automation tab 60
    - Channel comment 547
    - Channel types 35
    - Channels tab 34
    - Contents of file 66
    - Event channel 548
    - File name 547
    - Functions 325
    - Get information 545, 560
    - Limit file size 549
    - Limit sample time 550
    - Load from data file 140
    - Loading and saving 140
    - Mode 54
    - Mode tab 54
    - Number of channels 546
    - Number of repeats 552
    - Optimise rate settings 550
    - Play waveform tab 57, 59
    - Remove channel 546
    - Reset configuration 546

- Sampling configuration 34, 35, 66, 272
  - Resolution tab 46
  - Sample bar label and comment 547
  - Sample mode 550
  - Save 432
  - Sequencer tab 56
  - Spike2 Last file 66
  - Talkers 44
  - TextMark data 41
  - Triggered start 556
  - Type of a channel 545
  - View handles 552
  - WaveMark data 43, 559, 560, 639
- Sampling Notes 273
- Save data channel 245
- Save files 60, 131
  - Automatic name generation 60
  - Automatic save of script 158
  - Automatically at end of sampling 60
  - during sampling 60
- Save resource from script 434
- Scale bar 176
- Scale using axis 12
- Scale waveform 375
- Scheduler 167
- SCLK sequencer directive 93
- Scope of variables and user-defined functions 312
- Screen dump
  - Interactive 146
  - Selected view area 145
  - To printer from script 431
  - Visible view area 145
  - Visible view area (script) 431
- Script
  - Automatic save if modified 158
  - Call stack 296
  - Debug 293, 395, 396, 397, 398
  - Enter debug on error 295
  - Handle of running script 329
  - Idle time control 167
  - Run script after current one ends 562
  - Size limits 315
- Script Bar
  - Control from script language 561
  - Description 281
  - Get handle 329
- Script language
  - Function index 329
  - Introduction to scripting 288
  - Syntax colouring 293
- Syntax index 296
- Text format 296
- Topics index 316
- Script menu 280
  - Compile script 280
  - Debug bar 281
  - Evaluate 280
  - Run script 280
  - Script Bar 281
  - Turn recording on and off 280
- Script window 292
  - Call tips 161
  - Clear all break points 292
  - Compile 292
  - Debug 294
  - Find func or proc 292
  - Read only scripts 292
  - Run 292
  - Set and clear break points 292
- script.tip 160
- ScriptBar() 561
- ScriptRun() 562
- Scripts
  - getting started 766
  - why use them 766
- Scroll bar
  - show and hide 613
- Scroll bar show and hide 176
- Scroll display 413
- Scroll using axis 12
- Scrolling while sampling 204
- SD (Standard Deviation) 195
- SDAC sequencer directive 93
- SE\_INC\_BASE\_PRIORITY\_NAME privilege 767
- Search data 262, 263
  - Active cursors 389
  - For feature 375
  - Start cursor search 393
  - Test search 394
- Search for text 154
  - and replace it 155
- Search for view 601
- Seconds() 562
- Select a channel 376
- Select all 153
- Select all copyable items 418
- Selection 15
- Selection margin 203
- Selection\$() 563
- SEM (Standard Error of the Mean) 195
- Semicolon
  - statement separator 304
- Send Mail 142
- Sequence step numbers 83
- Sequencer 70
  - See: Output sequencer 70
- Sequencer control panel handle 329
- Serial line
  - Conditioner connections 731
  - For 1902 166
  - Script commands 328
  - SerialClose() 563
  - SerialCount() 563
  - SerialOpen() 563
  - SerialRead() 564
  - SerialWrite() 565
  - Signal conditioner 166
  - TextMark data 41, 557
- Serial number 285
  - Read 329
- SerialClose() 563
- SerialCount() 563
- SerialOpen() 563
- SerialRead() 564
- SerialWrite() 565
- Set Marker Codes 252
- SET sequencer directive 93
- SetAverage() 566
- SetEvtCrl() 566
- SetEvtCrlShift() 567
- SetINTH() 567
- SetPhase() 568
- SetPower() 568
- SetPSTH() 569
- SetResult() 569
- SetWaveCrl() 570
- SetWaveCrlDC() 570
- Shell extensions
  - Remove 30
  - SonCols 30
  - SonInfo 30
- Short cut keys
  - Data views 25
  - Script bar 282
  - Text views 26
- Short interval between clustered events 696
- Show channel (list) 376
- Show hidden window 256
- Show line numbers 204
- Show window 256, 610
- Show X axis scroll bar 176, 613

- Show y axis 622
- Shuffled event correlation 567
- Sigmoid fit 361, 444
- Sigmoid fitting 457
- Signal conditioner 381
  - CED 1902 728
  - CED1902 731
  - Connections 731
  - Control panel 728
  - CyberAmp 728, 731
  - Digitimer D360 731
  - Get and set gain 383
  - Get and set offset 385
  - Get and set special features 381
  - Get list of gains 384
  - Get list of sources 387
  - Get offset range 385
  - Get revision 385
  - Get type 387
  - List filter frequencies 382
  - List filter types 383
  - Low-pass and high-pass filters 382
  - Online chanes 730
  - Power1401 gain option 728, 731
  - Read all port settings 384
  - Sample menu 273
  - Script commands 326
  - Set all parameters 386
  - Set gain and offset 730
- Sin() 571
  - As event kernel 244
  - Virtual channel function 237, 238
- Sine of an angle in radians 571
- Sine wave output 102
- Single step a script 294
- Sinh() 571
- Sinusoid fit 361, 444
- Sinusoidal fitting 459
- Site licence 4
- Size of window 609
- Skewness 337
- Skip NaN 247
- Skyline display mode
  - Result views 195
  - Time views 194
- Slope
  - Between cursors 266
  - ChanMeasure() 367
  - trend plot 220
- Slope of wave 247
- Slope peak and trough cursor modes 263
- Slope search cursor modes 263
- Slope% cursor mode 263
- Slow channels 46, 52
- Slow response in Sampling configuration 52
- Slow response sampling with 1401 ISA interface 769
- Slow scrolling with AGP card 771
- SMControl() 571
- Smooth wave 247
- Smoothed frequency 419
- Smoothing of arrays 334
- SMOpen() 572
- smr standard file extension 122
- Software help desk 284
- Software Licence 4
- Solid colour 518
- Solve linear equations 496
- SON data file versions 20
- SonCols 30
- SonFix data recovery utility 737
  - Batch processing 738
  - Manual fix 739
  - SonFix options 738
  - Using SonFix 737
- SonInfo 30
- Sonogram
  - Colours 202
  - Display details 191
  - Drawing mode 191
  - Drawing mode (script) 413
  - Key 192
  - Zero dB 193
- Sonogram display mode 191
- sonview.tip file 284
- Sorting
  - Arrays 336
  - Channels 369
  - Raster display 195
- Sound card 275, 646
- Sound output 572
- Sound() 572
- Sources of information 17
- Spawn program 534
- SpE() virtual channel function 235, 240
- Speak() 573
- Spectral edge 235, 240, 241
- Speech output 573
- Spike monitor window
  - Control state from script 571
  - Interactive use 662
- Open and get handle 572
- View menu command 198
- Spike shape window
  - Apply collision analysis 576
  - Button states 575
  - Collision analysis important area 577
  - Collision analysis information 578
  - Create new spike channel 576
  - Delete templates 580
  - Display range 624, 625
  - Duplicate channels 360
  - Get and set template information 581
  - Get template and display size 582
  - Get template data 580
  - Merge or replace template 582
  - Open dialog 579
  - Optimise display 516
  - Parameters 579
  - Reclassify channel 576
  - Run state 580
  - Script function list 327
  - Set channel 575
  - Set template and display size 583
  - Set template code 646
  - Set trigger levels 466
  - SS...() commands 574
- Spike sorting 638
  - Clustering 676
  - Create one channel per template code 658
  - Detecting spikes 638
  - Dialogs 640
  - Digital output on match 639
  - Horizontal cursors 644
  - Link cursors 644
  - Load templates 660
  - Off-line editing 655
  - Off-line sorting 652
  - On-line display 659
  - Peak between cursors mode 652
  - Sampling rate 639
  - Setup sampling for 43
  - Template controls 646
  - Template formation 649
  - Template setup 642
  - Topics 638
- Spike2
  - Changes and revisions 759
  - Command line 29
  - Icons 2
  - Introduction 2

- Spike2
  - Manuals 17
  - Removing from your system 5
  - Updating from the web site 5
- Spike2 Last configuration file 66
- Spike2 top box 38
- Spike2 versions 5
- Spikes per second
  - As a waveform 234
  - Event correlation 208
  - Instantaneous frequency drawing mode 189
  - Mean frequency drawing mode 189
  - PSTH 207
  - Rate drawing mode 189
- Spreadsheet output 148
- Sqr()
  - Virtual channel function 238
- Sqr() virtual channel function 237
- Sqrt() 574
  - Virtual channel function 237, 238
- Square root 574
- srf Result file extension 122
- SS...() overview 574
- SSButton() 575
- SSChan() 575
- SSClassify() 576
- SSColApply() 576
- SSColArea() 577
- SSColInfo() 578
- SSOpen() 579
- SSParam() 579
- SSRun() 580
- SSTempDelete() 580
- SSTempGet() 580
- SSTempInfo() 581
- SSTempSet() 582
- SSTempSizeGet() 582
- SSTempSizeSet() 583
- Stability
  - of IIR filter 471
- Standard deviation
  - between cursors 266
  - Burst statistics 354
  - Curve fitting 226, 628
  - Error bars 349, 413
  - In result view 195
  - Measure region 367
  - of array 339
  - trend plot 220
- Standard display settings 176
- Standard error of the mean 195
  - Error bars 349, 413
- Standard settings for a view 605
- Start sampling 64
- startup.s2s 29
- State display mode 194
- Statements 304
  - format 296
- Static cursor 263
- Statistical distributions
  - Binomial 341
  - Chi-Squared 464
  - F distribution 346
  - Normal 464
  - Poisson 464
  - Student's 342
- Status bar 329
  - Description 172
  - Handle 329
  - Show and hide 329
- Stereotrode
  - Display 648
  - Non-sequential ports 43
  - Non-sequential ports dialog 44
  - Setup in WaveMark dialog 43
- sTick() sequencer expression 91
- Stop band 718
- Stop Process command 215
- Stop sampling 64
- Str\$( ) 583
- Straight line fit 361, 450
- String
  - Remove leading and trailing white space 595
  - Remove leading white space 595
  - Remove trailing white space 595
- String input 480
- Strings 299, 480
  - Arrays of 300
  - ASCII code 340
  - Constant 300
  - Conversions 324
  - Convert a number to a string 583
  - Convert ASCII code to string 379
  - Convert to a number 599
  - Convert to lower case 482
  - Convert to upper case 598
  - Currently selected text 563
  - Delete substring 399
  - Extract fields from 541
  - Extract fields setup 541
  - Extract middle of a string 507
- Find string within another string 480
- Get rightmost characters 542
- Leftmost characters of string 483
- Length of a string 483
- Printing into 529
- Read from binary file 351
- Read string from user 480
- Reading using a dialog 411
- Script functions 324
- Specifying legal characters in input 480
- Variable 299
- Write to binary file 356
- Student's distribution 342
- SUB output instruction 112
- Sub-array 300
- Substring of a string 507
- Subtraction of arrays and values 338, 339
- Sum
  - Measure region 367
  - of array 339
  - of array product 332
  - of channels 232
  - of result view bins 266
  - of Result view bins (script) 388
- Sweeps in a histogram 179
- Sweeps() 583
- sxy XY file extension 122
- Syntax colouring 293
- Syntax of script language 296
- System handle count 329
- System slow when scrolling 771
- System toolbar 172
- System\$( ) 584
- System() 584
- SZ output instruction 102
- SZINC output instruction 103

**- T -**

- TABADD output instruction 114
- TABDAT sequencer directive 94
- TABINC sequencer directive 114
- TABLD output instruction 114
- TabPos() sequencer expression 91
- TabSettings() 585
- TABST output instruction 114
- TABSUB output instruction 114
- TABSZ sequencer directive 94
- Talker
  - Duplicate channel 34

- TalkerRead() 585
- Talkers
  - Add Talker channel dialog 45
  - Configure dialog 45
  - Create a talker-based channel 556
  - Disconnect a Talker 273
  - Documentation 45
  - Examples 46
  - Forget a Talker 273
  - Information 273
  - Items 45
  - Read back information 585
  - Sampling configuration 44
  - Send string to a talker 586
- TalkerSend() 586
- Tan() 586
  - Virtual channel function 237, 238
- Tangent of an angle in radians 586
- Tanh() 587
- Technical support 758
- Template 660
  - Cluster on correlations 682
  - Cluster on errors 683
  - Copy to clipboard 653
  - Dialog 650
  - Digital output on match 639
  - Edit code 646
  - Editing 655
  - Formation algorithm 649
  - Load 660
  - Merging 646
  - Off-line formation 652
  - On-line/Off-line 659, 660, 661
  - Peak between cursors mode 652
  - Print templates 653
  - Setting spike region 643
  - Setup window 642
  - Topics 638
- Ternary operator
  - In dialogs 23
  - In script language 306
- Test program for 1401 741
- Test programs 734
- Tetrode
  - Display 648
  - Non-sequential ports 43
  - Non-sequential ports dialog 44
  - Setup in WaveMark dialog 43
- Text caret
  - Get and set position 515
  - Get column number 515
  - Get line number 515
  - Get position and set relative 515
  - Multiple selections 28
  - Scroll view 413
- Text copy 416, 417
- Text display mode 195
- Text file
  - Create (script) 427
  - Formatted text output 527
  - Open (interactive) 123
  - Open (script) 428
  - Read 539
  - Save (interactive) 131
  - Script commands 321
- Text from different system 153
- Text import default format file 124
- Text output
  - As clipboard text (result view) 149
  - As clipboard text (spreadsheet) 148
  - As clipboard text (time view) 150
  - As clipboard text (XY view) 149
  - From result view 149
  - From time view 150
  - From XY view 149
- Text to speech 573
- Text view 515
  - Caret colour 160
  - Default style 160
  - Folding margin 203
  - Font size 27
  - Force upper/lower case 27
  - Get bottom line 612
  - Get column number 515
  - Get line number 515
  - Get top line 612
  - Gutter 203
  - Line numbers 204, 602
  - Maximum lines 604
  - Modified 513
  - Move absolute 515
  - Move relative 515
  - Move to line number 515
  - Read only 513
  - Script commands 319
  - Scroll view 413
  - Selection margin 203
  - Set top line 413
  - Short cut keys 26
  - Tab settings 585
  - Tab size 160
  - Zoom text (interactive) 27
  - Zoom text (script) 607
- TextMark data 274
  - Add marker from script on-line 557
  - Add to memory channel 506
  - Channel types 21
  - Copy As Text format 151
  - Copy to clipboard 274
  - Create channel for sampling 557
  - Create during sampling 274
  - Display text vertically 177
  - Edit existing mark 488
  - Edit TextMark dialog 274
  - Enable for sampling 41
  - Export As format 151
  - Get information 489
  - Jump to marker in list 274
  - Read string 482, 516
  - Serial line input 41
  - Text display mode 195
  - Vertical marker 177
- then
  - in case statement 307
  - in if statement 307
- Threads 167
- Threshold crossing cursor modes 263
- TICK0 output instruction 116
- TICKS output instruction 116
- Tile windows 256
- Time 482, 516
  - Change data file creation 437
  - data file creation 435, 436
  - In automatic file name 60
  - Into sampling 497
  - Maximum time in a file 497
  - Resolution for channels 350
- Time at point 220
- Time base adjustment 435
- Time difference 220
- Time expressions in dialogs 23
- Time limits 60
- Time of day 173
  - as a string 587
  - as numbers 587
  - in a data file as string 435
- Time resolution 20, 46
- Time shift 334, 717
  - Channel of data 373
- Time shift wave 247
- Time units per ADC convert 46, 52
- Time view
  - Apply resource file 423
  - Background colour 380
  - Background palette colour 379

- Time view
    - Convert seconds to Spike2 time units 613
    - Copy as Text 148, 150
    - Copy data to array 359
    - Count of events 388
    - Duplicate 256
    - Modified 513
    - Of next item on a channel 516
    - Of previous item on a channel 482
    - Printing 430
    - Process data 530
    - Save resource file 434
    - Script commands 318
    - Time of next item 516
    - Time of previous item 482
    - Value at given time 377
  - Time zero 265
  - Time\$( ) 587
  - Timed sampling mode 54
  - TimeDate() 587
  - Timing built-in functions 397
  - Timing faults in the graphical editor 76
  - Tip of the Day 284
  - Tips in script views 161
  - Title of window 610
  - Title string for channel 377
  - Toggle comments 156
  - Toolbar 329, 588
    - Add buttons 593
    - Change text 594
    - Clear all buttons 589
    - Enable and disable buttons 589
    - Get last button 593
    - Idle function 588, 593
    - Play waveform 57
    - Show and hide 594
    - System toolbar 172, 329
    - window handle 329
  - Toolbar() 588
  - ToolbarClear() 589
  - ToolbarEnable() 589
  - ToolbarMouse() 590
    - Example 593
  - ToolbarSet() 593
  - ToolbarText() 594
  - ToolbarVisible() 594
  - Tooltips
    - DlgButton 402
    - In user-defined dialog 399
    - Interact 481
    - Toolbar 593
  - Topics script index 316
  - Trace of matrix 497
  - Trace through a script 294
  - trans() 300
  - Transpose of matrix 497
  - Trend plot
    - MeasureChan() 498
    - MeasureToXY() 499
    - MeasureX() 500
    - MeasureY() 501
  - Trend plot example 218
  - Triangle
    - As event kernel 244
  - Trigger
    - Rear panel 165
  - Trigger channel 208
    - for correlation 208
    - for correlations 208
    - for Phase histogram 210
    - for PSTH 207
    - for waveform average 211
  - Trigger connections 57
  - Trigger/Overdraw menu command 182
  - Triggered displays 183
  - Triggered processing 769
  - Triggered sampling mode 54
    - Keyboard marker trigger 40
    - TextMark channel as trigger 274
  - Triggered start of sampling 64, 555, 556
  - Triggered time view 606
  - Triggered waveform output 57
  - Trim() 595
  - TrimLeft() 595
  - TrimRght() 595
  - Troubleshooting 741
  - Trough find cursor mode 263
  - Trunc() 596
  - Truncate real number 596
  - Try1401 284
  - Try1432.exe 741
  - t-test 342
  - TTL compatible signals 39
  - Turning point cursor mode 263
  - Tutorial 8
  - Two band pass filter 712
  - Two band stop filter 712
  - txt text file extension 122
  - Txt\_Def.cim 124
  - Type compatibility 299
  - Types of data 297
- U -
- U1401Close() 596
  - U1401Ld() 596
  - U1401Open() 597
  - U1401Read() 597
  - U1401To1401() 597
  - U1401ToHost() 598
  - U1401Write() 598
  - UCase\$( ) 598
  - Underlying time units 20
  - Undo command 148
  - Units for waveform data 36
  - Units for waveform or WaveMark channel 377
  - until 308
  - Update all views 413
  - Update invalid regions in a view 413
  - Upper case a string 598
  - us as time scaler 23
  - us() sequencer expression 91
  - USB vs PCI interface 769
  - User data path 31
  - User interaction
    - Ask user a Yes/No question 535
    - Command summary 321
    - Create dialog 405
    - Create Toolbar 588
    - Dialogs 399
    - From script 405, 481, 588
    - Input single key 479
    - Message in pop-up window 507
    - Print formatted text 527, 529
    - Read a number in a pop-up window 479
    - Read a string in a pop-up window 480
    - Test for key available to read 481
    - The toolbar 588
  - User-defined functions and procedures 310
  - usTick() sequencer expression 91
  - Utility programs 734
    - S64Fix 734
    - SonFix 737
    - Try1401 741
- V -
- Val() 599
  - Valid cursors 262
  - Value above baseline 220, 500
  - Value at cursor 265



- Value at point 220
  - Value between cursors 266
  - Value difference 220
  - Value parameters 311
  - VAngle() sequencer expression 91
  - var 299
    - array declaration 300
  - VAR sequencer directive 92
  - Variable
    - Inspecting value 295
    - Names 297
    - Types 297
  - Variable declarations 299
  - Variables for output sequencer 92
  - Variance 337
    - Curve fitting 628
  - VarValue script 92, 97, 98, 99, 101, 106
  - VDAC0-7 sequencer variables 92
  - VDAC16() sequencer expression 91
  - VDAC32() sequencer expression 91
  - VDigIn sequencer variable 92
  - Vector 300
  - Versions of Spike2 5, 759
  - Vertical bar notation 291
  - Vertical cursor
    - Drag back into view 10
  - Vertical cursor commands 320
  - Vertical Marker dialog 177
  - Vertical Markers
    - Get Font 462
    - Interactive control 177
    - Line thickness 163
    - Script control 599
    - Set Font from script 462
  - Vertical space for channels 378
  - VerticalMark() 599
  - VHz() sequencer expression 91
  - Video codecs 751
  - Video frame rate 749
  - Video recording 746
  - View handle 289, 600
    - Close view 423
    - Create result view 565
    - Cursor dialogs 392
    - Duplicate views 415
    - File name 427
    - Find from view title 601
    - for Log window 488
    - for new result view 566, 567, 568, 569, 570
    - for new view 427
    - for opened view 428
    - for sampling windows 548
    - for system windows 329
    - Get and set 600
    - Get colour for view 601
    - Get linked views 602
    - Get list of views 603
    - Get type of view 602
    - Interactive access to 257
    - Override current view 600
    - Sampling windows 548
    - Set colour for view 601
    - Set or get current view 600
    - Update the view 413
  - View manipulation functions 316
  - View menu 172
    - Channel Draw Mode 188
    - Colour commands 199
    - Display trigger 183
    - Enlarge and reduce 172
    - File information for time view 179
    - Font 198
    - Info 179
    - Overdraw 3D 186
    - Overdraw List 185
    - Rerun 204
    - Result view drawing modes 195
    - Show/Hide channel 176
    - Spike Monitor 198
    - Standard display 176
    - Trigger/Overdraw 182
    - X Axis Range 173
    - XY options 182
    - Y Axis Range 174
  - View() 600
  - View().x() 600
  - View-based expressions 23
  - ViewColour() - deprecated 600
  - ViewColourGet() 601
  - ViewColourSet() 601
  - ViewFind() 601
  - ViewKind() 602
  - ViewLineNumbers() 602
  - ViewLink() 602
  - ViewList() 603
  - ViewMaxLines() 604
  - ViewOverdraw() 604
  - ViewOverdraw3D() 605
  - ViewStandard() 605
  - ViewTrigger() 606
  - ViewUseColour() 607
  - ViewZoom() 607
  - Virtual channels 232
    - Arithmetical operators 234
    - Build expression interactively 238
    - Channel functions 234
    - Comparison operators 234
    - Event to waveform 236
    - Mathematical functions 237
    - Operators in expressions 234
    - Spectral functions 235
    - Waveform generation 237
  - Virtual space in text 28
  - VirtualChan() 607
  - Visible state of a window 610
  - Visible state of channel 378
  - Voltage limits
    - 10 Volt/5 Volt ADC inputs 165
    - TTL inputs 39
  - Voltage range for 1401 ADC/DAC 165
  - VSz() sequencer expression 91
- W -**
- Wait in script 624
  - WAIT output instruction 99
    - Clash with template match 639
  - WAITC output instruction 105
  - Watch window 295
    - Debug 294
  - Waterfall script example 632
  - WAVE file output 572
  - WAVEBR output instruction 120
  - Waveform calibration 245
  - Waveform data 21, 36
    - 10 Volt/5 Volt range 165
    - accumulate or copy to result view 566
    - Aliasing 36
    - Amplitude 377
    - Average 211
    - Calibrate 245
    - Channel dialog 35
    - Channel types 21
    - Convert to events 230, 652
    - Convert to Marker 652
    - Convert to WaveMark 652
    - Copy As Text and Export As format 151
    - Copy into an array 359
    - Correlation 570
    - Create from events 419
    - Display mode 194
    - Drawing colour 380
    - Fill gaps 247

- Waveform data 21, 36
    - Generate 232, 237
    - Level crossing 230
    - Mean level 388
    - Offset 369
    - Output during sampling 326
    - Palette colour 379
    - Peak search 230
    - Power spectrum 212, 568
    - Sample rate 21, 36
    - Sampling interval 350
    - Sampling setup 559
    - Scaling 21, 36, 375
    - Units 377
    - Virtual channels 232, 237
    - Waveform channels 36
    - Write to channel 379
  - Waveform output 275
    - Add waveform to list 520
    - Change and get DAC list 522
    - Change area size 525
    - Change cycles 523
    - Control bar buttons 524
    - Delete area 523
    - During sampling 70
    - Enable area 523
    - Get area information 523
    - Link and unlink areas 524
    - On-line 57
    - Output rate variation 526
    - Play offline 519
    - Sample rate 525
    - Secondary key code 524
    - start and stop from sequencer 120
    - Status during output 526
    - Stop output 526
    - test from output sequencer 120
    - Trigger state 527
    - Update waveform on-line 522
  - WAVEGO output instruction 119
  - WaveMark data 190, 638
    - Channel types 21
    - Convert to waveform 234
    - Copy As Text and Export As format 151
    - Detecting spikes 638
    - Display mode 194
    - Drawing colour 199, 380
    - Drawing colour (script) 380
    - Editing 655
    - From waveform 652
    - Get information 489
    - Locate in overdraw mode 190
    - Monitor multiple spike channels 662
    - Non-sequential ports dialog 44
    - Offset 369
    - Overdraw display mode 190
    - Overdraw mode 413
    - Palette colour 379
    - Reclassify 655
    - Sampling interval 350
    - Scaling 375
    - Script control of template windows 327
    - Set codes using the Mouse 190, 194
    - Setup for sampling 560
    - Setup sampling for 43
    - Show setup dialog 427
    - Spike sorting setup 639
    - Traces 228, 229, 368, 482, 488, 489, 501, 503, 506, 516
    - Units 377
  - WAVEST output instruction 120
  - Web page 758
  - Weighting of channel space 378
  - wend 308
  - WEnv() virtual channel function 237
  - while 308
  - Window data for FFT 332
  - Window duplicate 256
  - Window for FFT 212
  - Window menu 256
    - Arrange icons 257
    - Cascade 256
    - Close All 257
    - Close All and Link 257
    - Hide 256
    - Show 256
    - Tile 256
    - Windows 257
  - Window() 608
  - WindowDuplicate() 609
  - WindowGetPos() 609
  - Windows
    - Close window 423
    - Current view 600
    - Duplication 609
    - Get linked view 602
    - Get list of view handles 603
    - Help 469
    - Manipulation functions 316
    - Number input with prompt 479
    - Pop-up message window 507
    - Position 608, 609
    - Query user in pop-up window 535
    - Show and hide 610
    - Standard settings 605
    - String input with prompt 480
    - Title 610
    - View handle 600
  - Windows dialog 257
  - Windows versions 766
  - WindowSize() 609
  - WindowTitle\$() 610
  - WindowVisible() 610
  - Working Set 531
    - Full explanation 767
    - In About Spike2 dialog 285
  - World Wide Web 758
  - WPoly() virtual channel function 237
  - Write binary data 354, 356
  - Write memory buffer to channel 231
  - Write memory channel to disk 505
  - Write to disk 54, 64
  - WSin() virtual channel function 237
  - WSqu() virtual channel function 237
  - WT() virtual channel function 237
  - WTri() virtual channel function 237
  - WWW 758
- X -**
- X axis
    - Auto-units (script) 611
    - Bin number conversions 351
    - Display set region 413
    - Drawing mode 611
    - Drawing style 611
    - hms and time of day 611
    - Increment per bin in result view 350
    - Left hand value 612
    - Range 612
    - Right hand value 612
    - Scale bar 176
    - Scroll bar show/hide 176
    - Show and hide features 611
    - Show and hide scroll bar 613
    - Tick spacing 611
    - Title 613
    - Units 613
    - Value at given position 377
  - X axis control
    - Auto units 173
    - milliseconds 173
    - Mouse wheel 173
    - Scroll 173
    - Short cut keys 172, 173
    - Tick spacing 173

- 
- X axis control
    - Zero at trigger 183
  - X Range dialog 173
  - XAxis() 610
  - XAxisAttrib() 611
  - XAxisMode() 611
  - XAxisStyle() 611
  - XHigh() 612
  - XLow() 612
  - xor 304
  - XOR output instruction 113
  - XORI output instruction 113
  - XRange() 612
  - XScroller() 613
  - XTitle\$() 613
  - XToBin() 613
  - XUnits\$() 613
  - XY Draw Mode 196
  - XY view 182
    - Add data 614
    - Auto-expand axes 182
    - Automatic axis expansion 615
    - Background bitmap 363
    - Background colour 380
    - Background palette colour 379
    - Channel list 366
    - Channel offset 619
    - Channel offsets 634
    - Copy as text 149
    - Create a new channel 620
    - Create new XY view 427
    - Data joining method 618
    - Data range 620
    - Delete data points 615
    - Draw mode 196
    - Drawing styles 615
    - Fill colour 614
    - Fill with colour 196, 199
    - Get data points 616
    - Get or set title 377
    - Get or set units 377
    - Key 182
    - Limit points per channel 621
    - Line style 196
    - Modified 513
    - Modify all channel settings 620
    - Open file from script 428
    - Options 182
    - Overdraw data 632
    - Point style 196
    - Points inside a circle 617
    - Points inside a rectangle 618
    - Points inside a shape 616
    - Prompt to save unsaved view 158
    - Script commands 319
    - Script example 632
    - Set channel colour 614
    - Set Key properties 364, 619
    - Sort points 622
  - XYAddData() 614
  - XYColour() 614
  - XYCount() 615
  - XYDelete() 615
  - XYDrawMode() 615
  - XYGetData() 616
  - XYInChan() 616
  - XYInCircle() 617
  - XYInRect() 618
  - XYJoin() 618
  - XYKey() 619
  - XYOffset() 619
  - XYRange() 620
  - XYSetChan() 620
  - XYSize() 621
  - XYSort() 622
- Y -**
- Y axis
    - Auto-units 174
    - Auto-units (script) 622
    - Channel number show and hide 369
    - Drawing mode 623
    - Drawing style 622, 624
    - Get current limits 624, 625
    - Group channels 174
    - Horizontal title and units 176
    - Lock axes 174
    - Lock channels 623
    - No invert on drag 163
    - On right 176
    - Optimise 174
    - Overdraw 174
    - Range dialog 174
    - Range optimising 516
    - Right and left 623
    - Scale bar 176
    - Set limits (script) 625
    - Show all (script) 625
    - Show and hide 622
    - Show and hide features 623
    - Tick spacing 174
    - Tick spacing (script) 624
    - Title (result view) 180
    - Title (script) 377
    - Title (time view) 179
  - Y Range dialog 174
  - Y zero 265
  - YAxis() 622
  - YAxisAttrib() 622
  - YAxisLock() 623
  - YAxisMode() 623
  - YAxisStyle() 624
  - YHigh() 624
  - Yield time to the system 624
  - Yield() 624
  - YieldSystem() 625
  - YLow() 625
  - YRange() 625
- Z -**
- Zero dB for Sonogram 193
  - Zero region 266
  - Zero x axis at trigger 183
  - ZeroFind() 626
  - Zoom a channel 8, 11, 176
  - Zoom text
    - Remove zooming 605
    - Using keyboard or mouse wheel 27
    - ViewZoom() script command 607
-

