# Winamp In\_cwave plugin

This text has been written for the version 2.0.0 of plugin. Latter (now we have in\_cwave V2.2.2) we made some essential fixes here, but some part of the text still looks obsolete.

## 1 About in\_cwave

In\_cwave is WinAmp audio player input plugin. Historically, its purpose was to decode complex-valued audio files (CWAVE) for both playback and transcode modes of the player. It contains the universal quadrature modulator that is based on simple, but very powerful concept of user-defined DSP-list — arbitrary graph with unlimited ability to combine different modulation cells together with linear operations between audio-channels and intermediate complex audio values. All DSP processes are controlled by simple GUI-window in real-time during playback, so user can hear the result immediately.

In addition, the plugin has embedded real-time real-to-analytic signal converter, based on half-band low-pass filters. This provides facility to use any uncompressed mono / stereo WAV-files as plugin input files — the single limitation is low (below 48 KHz) sampling frequency of such files. (Higher frequencies are allowed, but the transformer wills cut-off lower frequencies.) Embedded analytic converter ensures sufficient quality for produced Hilbert-conjugate signal — much better than *most* of Hilbert transformers, which you can find in popular audio editors / libraries. So, at least for the first time you don’t need any CWAVE-files or CWAVE-converters. We know about 6-7 different ways to produce analytic signal from the real one, but only few of them look good enough to be used for sound. When we initially wrote embedded analytic signal converted, our feeling about was “Having a 10-point score for analytic signal quality, we will get seven for our embedded converter (dumb DFT-based converter, which consumes lots of memory, has 8).Surely, it is not bad”. This was in 2017 year. Now (2020), we are inclined to count the embedded converter slightly better, than DFT-based, especially for heavy compressed (most of available) sound. There was an idea to refuse from external CWAVE converter at all and make DSP-plugin, but DSP API of Winamp (and many other players too) looks too restrictive for our needs. At last, CWAVE format keep place to some experiments.

Besides WinAmp, the plugin sufficiently works with XMPlay audio player, which has an embedded support of WinAmp input plugins. Note that (by contrast to WinAmp) XMPlay is fully functional under Linux via Wine — together with in\_cwave.

In\_cwave also provides ability to produce 16- and 24-bits output together with rather rich control of dithering, noise shaping and quantization. In addition, we have additional feature of monitoring numerical stability for high-Q IIR filters and noise shaping.

Note that beside playback, our plugin also support transcoding audio-data to conventional audio-file, which can be used stand-alone.

The user should understand that in\_cwave is laboratory tool for experimental works — it is not intended to be consumer-used in a way like “simple-support-xxx-format-plugin”. Particularly, we suppose that our users have enough skills to work with tiny players setting as well as with analytical signal conversions. For many years, in\_cwave works as some sort of “handy and easy complex-sound breadboard” for us; in this role, it can be useful for you.

## 2 Plugin installation

First, you must compile the plugin. We provide two sets of projects to build it — for Microsoft Visual Studio 2013 and for Code::Blocks V16.01 with MinGW gcc V4.9.2, x86/32 bit. (Also tested Code::Blocks 17.12/ MinGW gcc V5.1.0; more recent version may require some fixes, see source code.) Note, that in both cases you have two projects — to build the plugin (in\_cwave.dll) and to build some small test program (test\_mt\_jrnd.exe) which is a part of our edition of Mersenne Twister pseudorandom number generator (we are using Mersenne Twister for dithering needs — in general you don’t need the test program). In general, all you need is a release version of in\_cwave.dll; in all cases, no additional (redistributable) components required.

In addition, you have to obtain and install WinAmp and / or XMPlay player(s). We do not know about compatibility with early versions of these players, so please use WinAmp V5.666 (the last 5-th version) and / or XMPlay V3.8.2.3. (Our plugin is a “Unicode plugin” in WinAmp sense.) Note, that with another Winamp version you can get problems.

Before describing the installation and corresponding player setup, we are to make some general notes about the players. To make choice note, that you can install in\_cwave in both players, and in this case, it will share the common configuration file, so you can switch between players according to you needs. Both of the players are free, WinAmp is not so big program, and XMPlay is very small and light-weighted.

### 2.1 WinAmp vs. XMPlay in in\_cwave context

*NOTE: this comparison relates only to the very specific bundle (player + in\_cwave plugin) — not to players in general.*

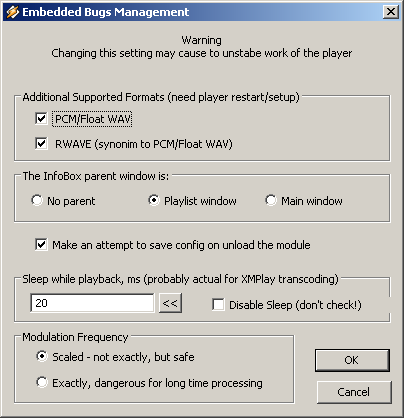
* In\_cwave is a native plugin for WinAmp; so with XMPlay, which have very different native API, you can meet unpredictable problems. As example, when you attempt to playback a file with 192 KHz sampling frequency AND you use WASAPI output in XMPlay — you get very strange sound in all cases other than “output resolution — 16 bits”; standard output audio interface also works well. WinAmp works correctly here.
* Under Linux Wine Windows emulator, emulation XMPlay is completely functional; but WinAmp is not. Particularly, while running under Linux Wine (we are using Open Suse 13.1) WinAmp has no menu topic “Send To” for transcoded the files; many other functionalities of WinAmp under Linux, including “Setting” window, do not look robust.
* XMPlay windows system looks more convenient while our setting / control window is open; WinAmp’s windows lose a part of functionality, while our control window is open.
* WinAmp supports batch processing for file transcoding; XMPlay provide per-file transcoding or transcoding as “playback to files”, which did not look very conventionally. WinAmp makes transcoding faster than XMPlay, and has ability to playback during transcoding; XMPlay stops playback when transcoding of file starts.
* WinAmp fails to support transcoding to 24-bit FLAC format; when you want to transcode in\_cwave output to a file with 24-bit resolution — the only way is to save it to WAV; in contrast XMPlay correctly save 24-bit output to FLAC files, but in WAV format it keeps only 16-bit sound, i.e. makes some post-processing.
* WinAmp chose input plugin only by input file extension, but XMPlay has embedded support for some basic formats, e.g. WAV, which identifies itself by content of the file header. This makes some additional difficulties to route WAV files to our plugin.

### 2.2 Installation and setup of in\_cwave with WinAmp

Install WinAmp. If you plan to use embedded in\_cwave analytic signal transformer with WAV files, turn off WinAmp’s native WAV-file support. To do this, open WinAmp “Preferences”; select “Plug-ins”/”Input”/”Nullsoft Waveform Decoder”/”Configure”. In window “Waveform Decoder” uncheck “wav”. Press “ok”, “Close”; exit WinAmp.

Copy our plugin binary “**in\_cwave.dll**” to WinAmp plugins folder **(…\Winamp\Plugins\** — “…” is installation path, usually “**C:\Program Files**” or “**C:\Program Files (x86)**”). Start WinAmp. Now we have to make some basic setup.

Open WinAmp. Open “Preferences”; select “Plug-ins”/”Input”. In the list “Input plug-ins”, you will see a record with our plug-in (“CWAVE/WAV (“RWAVE”) Decoder/Modulator V… — in\_cwave.dll”). If none, something goes wrong, probably, from compilation step. You can check “About” and see our basic version and copyright information. Now click “Configure”, and our main control window will appear. It may look complicated; we will describe it latter. For now, the only button makes interest for us: “Bugs control…” at the bottom-right corner. Click it. The following dialog will appear:



Picture 1. The system-setting window

The title “Bugs Management” relates to the different hacks / technics that we are using in our plugin while the conventional plugins do not. Here you should determine, what kind of input files you want to handle. If you are interested in ready to use complex audio wave files (CWAVE) only — please uncheck “PCM/Float WAV” and “RWAVE”. If you want to play uncompressed WAV files (with any bits’ resolution including floating), you can check “WAV” and / or “RWAVE”.

NOTE: “RWAVE” extension added as a synonym to “WAV”. This will help if you want to direct WAV support to the native lossless player plugin, while having an ability to play WAV files via our plugin. In this case, you may check “RWAVE” only and then rename corresponding xxx.WAV files to xxx.RWAVE, or create hard links to initial WAV’s.

NOTE: The WAV playback over our plugin is always “loss”: even you turn off any our DSP’s, you will hear the signal after analytic conversion system, which has made some signal transformations.

Other options of this window:

Parenting of plugin main control window (“The InfoBox parent window is:”): The safer choice is “Main Window”; the useful one is “No parent”. The choice “Playlist window” is somewhere between these two. Usually, it is useful to keep main control window of plugin permanently open. If you select “No parent”, you can freely use WinAmp transport control (main window), and partially playlist control (no add/remove files). If you select “Playlist window”, you can still use transport control, but for any operation with playlist, you are to close our window. In addition, if you select “Main window”, you cannot use transport functions while our window open, while interaction between our window and WinAmp windows will be almost correct. You can change your choice later.

The current state of our plugin is saved to separate file in the Application Data of home user folder. When the player terminates, it calls the special function (quit ()) for every plugin. This function intends to keep the current plugin state. The problem is that if our window is still open while the player terminates, the quit is not called and the current configuration will not save. To prevent the loss of the configuration (as far as it can be very complex) you can check “Make an attempt to save config on unload the module”. If it is checked, the plugin attempts to save an unsaved configuration while unloading in\_cwave.dll. It is rather safe in most cases, but we know that this is an “unclean solution”.

The next system parameter(s) is “Sleep while playback”. Generally — keep it default. The normal WinAmp value is 20 msec. If you set it to zero or check “Disable sleep”, then during the playback, the plugin will render the endless internal loop and 100% load for one logical CPU. Do not change, if unsure; for details — see source code / playback.c. The maximum value can be set to sleep — 100 msec.

The last parameter “Modulation Frequency” defines how plugin calculate the samples of modulation frequency. If you unsure, “Scaled – not exactly, but safe” will be right solution. For more precision information about — see section 3.1 “Some essential theory”.

After finished press OK, and, if the supported file types changed, close all setup windows and restart WinAmp. Now, you can make some additional setup of the player. Open “Preferences”. In “File Types”, make sure, that “CWAVE”, and your selected types (“WAV” and “RWAVE”) are supported. If your audio system supports 24-bit sound, in “Playback” set “Allow 24 bits”. Always uncheck “Allow surround”, “Use dither”, “Force Mono”. Turn off all internal sound DSP effects, including the equalizer. Check setting of output plugins to prevent uncontrolled sound transformation. We also highly recommend turning off the WinAmp’s option “Always on top” to prevent WinAmp’s windows meshing. Ok, now you are ready to work with our plugin.

### 2.3 Installation and setup in\_cwave with XMPlay

Refer carefully to WinAmp installation — the installation key points for XMPlay are rather the same as for WinAmp. So let’s take care on differences.

To install, place in\_cwave.dll into the folder used to install / unpack XMPlay (there are no separate subfolders for plugins here). Run XMPlay. Open “Options and stuff” window. Find “Plugins” / “Input”. In the list “Input and archive plugins” find record “Winamp [in\_cwave.dll] CWAVE/WAV (“RWAVE”) Decoder/Modulator…”. If none, something goes wrong. Check “About”. Press “Config”.

The most significant difference from WinAmp related to the embedded XMPlay mechanism to identify “well-known” audio files by format, not by filename extension. To make long story shorter, there are only two variants: to refuse WAV support in in\_cwave completely or to set only one file type to support WAV or its alias RWAVE. We recommend using WAV. After checking the “WAV” extension in in\_cwave “Bugs Control”, close all setup windows and restart XMPlay. Open again “Options and stuff” / “Plugins” / “Input”, select “Winamp [in\_cwave.dll]…”. Check, that in the list “Supported file types” the “cwave” and “wav” records are present. Switch to the field “Priority file types” and type “wav” in it. (Of course, you can use “rwave” type instead of “wav”; in this case, you also must set “rwave” as priority file type. Also, you can try to set both, but this configuration works in unpredictable manner — usually XMPlay plays via its own internal plugin, while showing in\_cwave window in “Plugin track info”.) On some cases, the possible problems can be eliminated by refusing from “rwave” type — use only “wav” instead.

Restart player to make sure that changes are accepted. Open “Options and stuff” / “Plugins” / “Output”. We recommend using standard Windows output device (not WASAPI). Set sample rate to any (recommended to “conventional”, i.e. 44100), “Channels” to “stereo”, “Resolution” to “24 bit”, uncheck all “Apply sample rate to all file formats”, “Down mix multi-channel”, “Dithering” and “Noise Shaping”. “SRC quality” set to maximum. In “File writing” subsection, check “Use source resolution”.

Then switch to “Output” / “Encoders”. Select “FLAC” as single possible lossless variant for XMPlay. Setup its command line by your taste. Make sure, that “Normalize” is unchecked, and “Resolution” set to “output”.

We also recommend setting up a shortcut for operation “List track — Plugin info” (we are traditionally using Alt+3 like the one embedded in WinAmp). This is a useful way to open our control window at any in\_cwave-handled track in playlist.

Close settings window — XMPlay is now ready to work with our plug-in.

Note: of course, the output settings may be differing from those recommended, but in this case, you may lose the control over the output signal. Our plugin makes much wider and clearly explained quantize / dithering / noise shaping / output resolution setting than any player used to experiment with it.

## 3 Working with plugin

### 3.1 Some essential theory

The key conception of the plugin is the ***DSP-list***, which is built from ***DSP-nodes*** or ***DSP-cells***. There are four types of DSP-nodes: Master, Spectrum Shift (Shift), Phase Modulation (PM) and Mixer (MIX). Each node defines elementary set of DSP-operations, which take a complex stereo signal from inputs and produce complex stereo signal to output. For each input complex stereo sample, all of the nodes in DSP-list executed sequentially — from bottom to top of the list. The general structure of DSP-node shown below:

Inputs: Input signal (In) and/or variables from A to Z

In

A

B

C

D

X

Y

Z

∑(selected inputs)

Left/Right channels exchanger

Left/Right channels Spectrum Inversion

Left/Right channels Gain Adjust

Left/Right channels Modulation with bypass

A

B

C

D

X

Y

Z

One of the output variables from A to Z

Picture 2. The generic DSP-node structure

As you can see, there are 27 virtual variables — one of them (called ‘In’) always contain current complex-valued two-cannel (stereo) sample value, which was read from the input file. (If the input file has only one channel, both left and right parts of ‘In’ are identical.) Other 26 variables, which signed by letters from ‘A’ to ‘Z’ contain or output values of previously executed node (if the one was connected to any output) or “digital silence”, i.e. zeros.

Execution of every DSP node begins from mixing (pure sum, w/o any weight coefficients) of selected input variables. Please note, that A-Z variables keep their values from previous execution of DSP-list, so you can organize some delay / simple digital filter(s), but we not sure that this is very useful. Note also that usually most (but not all!) of the “real” nodes has only one input variable.

After mixing summary signal go to channel exchanger, which can:

* Leave signal unchanged;
* Swap left and right channels;
* Set both channels to only left or only right channel;
* Set both channels to the mix to mono (mean value of left and right channels).

All of the subsequent stages of DSP-node can be executed independently for the left or right channels (or can be locked hard for both channels).

The next stage we call as “spectrum inversion”. It or leave the signal unchanged, or swap real and imaginary (In-phase and Quadrature, “I” and “Q”) parts of complex-valued signal. Our tip about — if you do not know for what it can be useful — keep the signal unchanged; we add the feature generally for algorithmic fullness.

The next stage — the gain adjusting. We multiply our signal to value from 0.0 to 2.0. If you need more (very rarely), you can use our “universal glue” — one or more sequentially coupled MIX DSP-node(s) with corresponding gain level. When you set the gain for the node, you shall not be afraid for local signal clipping — all of the computations have made in double floats and all you need is the resulting signal of the whole DSP-list don’t be clipped (and don’t be too low). So, for the most cases is enough to set the proper level on the Master node.

After gain adjusting we make modulation. Only Shift and PM DSP-nodes have the modulation stage; the Master and MIX node types have not modulation submodules. Take a point that every modulation-capable node has the instant bypass modulation switch — if the one is set to “on”, the node began to work as MIX-node.

Every modulation submodule makes the next operation:

Where S in/out[n] — complex-valued input and output samples; — modulation function, n — sample number; as usual j2 = –1. (We do not use here the “complex envelope” term while in the case is difficult to say about what is “envelope” by physical and another meaning(s). At some point of view both and have an equal rights in (1)).

The modulations function for Shift DSP computed by one of the following ways, depending of selection “Modulation Frequency” at picture 1:

— For mode “Exactly, dangerous for long time processing”:

— For mode “Scaled – not exactly, but safe”:

Where Fs — sample frequency (we assume that it is integer); Fm — the frequency shift in units of Fs. Note that Fm is signed number which positive values have given shift spectrum up, negative — down. Function *round(x)* return nearest integer to *x*. In\_cwave provide ability to set Fm from ‑20 to 20 Hz. The (2) is simple trivial linear phase of harmonic oscillator. It has obvious disadvantage — while sample number n unlimited growing, has unlimitedly growing too. So when it take too big value, there is a loss of significance in (1) occurs. While we compute (1) with double precision, we can accept some reasonable value for — for example, about 106. At Fs=44100 Hz and Fm=40 Hz maximum continuous playback time will be about seconds. Its look pretty enough, but we do not want to have so silly limits. As solution, we introduce (2) — it limit precision for Fm to 10-3 *by absolute value*, but prevent phase to grow bigger, than . We are sure, that there is a better than our solution exists, but we are lazy to find it. Sorry.

The modulation function for PM DSP-node looks slightly more complicate:

— For mode “Exactly, dangerous for long time processing”:

— For mode “Scaled – not exactly, but safe”:

Here Lpm — phase modulation level, can be set in bounds [0, 1]; Fp — phase modulation frequency, [0, 40Hz]; Pint — initial “internal” phase of modulation signal, [‑1, 1]; Pext — initial “external” phase modulation angle, [‑1, 1]. Note, that if you set Lpm=1, Fp=0 and Pint=0 you could simple change the *constant* phase shift for the entire signal phase from - to by changing Pext from ‑1 to 1. The difference between (4) and (5) is same as in case (2) and (3); note, that Fp can take only non-negative value.

We suppose that the two modulation functions (in meaning all of these linear and sequential combinations, which can be makes by our DSP-nodes / DSP-list) provide *algorithmically full set of acceptable modulations for non-noise sound*. By “acceptable”, we mean:

1. The modulation function must be continuously differentiable — this is obvious rule for instant phase / instant frequency for audio non-noise signal — we call this “***Rat condition***”.
2. The absolute value of derivative of modulation function (instant frequency) cannot unlimited grow in time — it must have hard limits; any another case brings the instant frequency of target signal out of any meaning. We call this “***Catcher condition***”.

Both conditions we call “***Rat and Catcher conditions***”. Of course, there are many function classes exists (like Bessel or functions, for example) which satisfy Rat and Catcher conditions, but we do not count such functions are useful to implement here. Therefore, we provide only (1) constant, (2) linear function and (3) harmonic function as well as all of their superposition — and in the meaning our modulator is algorithmically full. If you are doubt — please note, that this is our plugin, and here we set the rules.

After modulation stage (for the MIX node — after adjusting gain), the resulting signal placed into one of virtual variables A-Z for use as input for the next DSP-node (or even for using in the next sample-loop of DSP-list). Please note that our plugin does not check virtual links, which has created according to user’s usage of the variables — this provides almost unlimited flexibility of the conception.

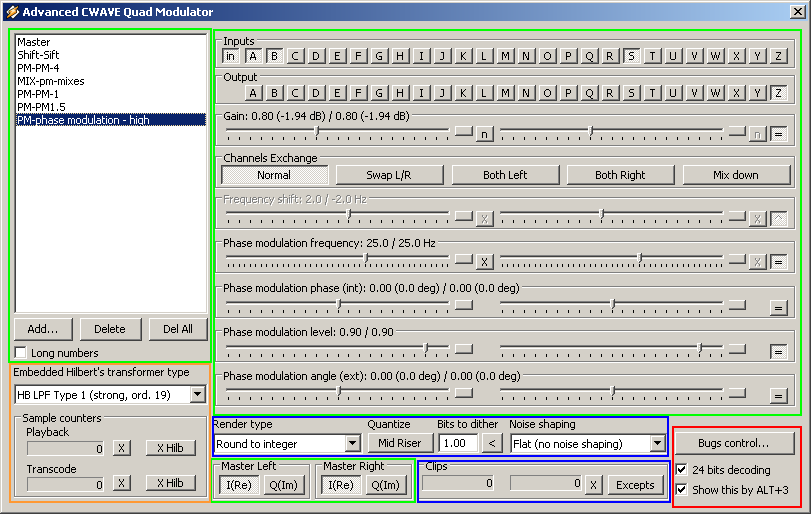
The Master node is very special. It always exists on the top of the list in one and only one instance. User cannot delete it or add to the list additional Master node(s). The functionality of Master node is almost same as MIX-node, but the Master does not place the result into any virtual variable — it places the result into per-channel I/Q selector. This provides to you the ability for per-channel choice between real and imaginary part for the *pure-real* result. After that, the real-valued signal came to the sound render, which makes optional dithering / noise shaping and quantization to “mastered” integer 16- or 24-bits sample values.

### 3.2 Main Control Window

While you made plugin installation and its basic setup, you already seen main control window calling it via player setup interface. While working, this window can be easy accessed via playlist context menu (“View file info” in WinAmp and “Plugin file info” in XMPlay). WinAmp has useful keyboard shortcut “Alt+3” to show this window (while a supported file — CWAVE / WAV / RWAVE is selected); the corresponding shortcut can be set also in XMPlay. Every time, when you want to close this window, you can simple press “Esc” key. Please note, that if you have multiply file selection in WinAmp’s playlist, the window will be sequentially open for each selected file — there is a property of WinAmp.

You should to understand, that every control in this window changes the plugin state immediately and the change have never related to any specific file — only to the current plugin state. While you exit a player, plugin has keep all of the setting, include current DSP-list in its config file (see section 2.2 for detail and possible problems); after a player run again, plugin restored last saved configuration.

So, let’s take a look to the main window more closely.



Picture 3. Main control window

For convenience, we split all of the controls to four groups:

1. Red — special controls;
2. Green — DSP-list controls;
3. Blue — sound rendering (dithering / quantize / noise shaping controls);
4. Amber — embedded analytic signal (“Hilbert’s”) converter.

So, let’s consider them in order.

#### 3.2.1 Special controls

This small group has only tree control, which behavior different from the others. You already know about button “Bugs control…” (See section 2.2 and picture 1). Sometimes is useful to change the control window parenting and very rare something else by this button.

Checkbox “24 bits decoding” related to sound render and determine resolution of the output of plugin. When unchecked, plugin produced 16-bits sound. As opposed to other controls in the window, the value of this control does not come to effect immediately — only when the next track starts to play or transcode. Note that if you have insufficient sound subsystem on your PC you can get troubles with 24-bits resolution.

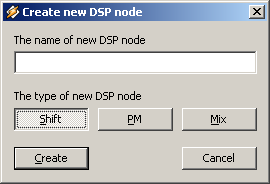
Checkbox “Show this by ALT+3” determine, which window will be shown by “View file info” / “Plugin file info”. If checked — show this window, if unchecked — show “normal” file info window, which contain information from CWAVE (WAV) file header and has additional ability to check audio data integrity for CWAVE-files with header version V2+. “Real” file info window also contains the checkbox “Show this by ALT+3”, which can toggle the type of window to show back. The value of this checkbox is a single parameter, which has not kept in config file — after player run, it set to show main control window.

#### 3.2.2 DSP-list controls

*This part is hardly based on section 3.1 materials.*

The left part of DSP-list control serves to manage the list in whole, i.e. to add, remove and select corresponding DSP nodes to inspect and edit, as well as to change shown format of the numbers.

The list box shows all DSP-nodes which now in DSP-list; the nodes executed from its bottom to top. At the top of the list always present one and only one Master node, as described in section 3.1. The button “Add…” launch the small dialog to add new node *into the bottom* of the list:



Picture 4. New DSP-node creation

Here you can type the human-readable name of new node, and select its type. The name will be shown in the DSP-list with a prefix, which identifies the node type. You cannot change the name and node type latter — if you make a mistake, you can only delete the node and create new one. The new node created with some defaults parameters, which you can freely change at any time. The bottom of DSP-list is only point, where you can create new node.

The button “Delete” active only if node selection stays at the last node in the list (the first node to process) and this node is not a Master. When you press it, the node will be silently deleted. “Delete All” button silently delete all of the nodes in DSP-list exclude Master regardless the node selector position in the DSP-list list box.

Please remember, that both nodes addition and deletion do not change the setting for the other nodes, which may prevent to unexpected effects to beginners. For example, if the last node in the list is the single one, which connected by input to in “In” virtual variable (likely case) and you has delete it, the sound will be interrupted up to you connect some existed node input to “In”. In general, nobody has check that DSP-list contains at least one true signal path from bottom cell to the Master as well as nobody has check, that some node output signal does not has a corresponding connection to any input. This is some payment for flexibility together with easy implementation. But more complicated implementation (with data flow graphs etc.) looks slightly comic for almost dead player’s plugin. It is completely another work, which need another design approach. In\_cwave serve to taste the conception(s) at all and we enforce to sacrifice some conveniences for ease and clean implementation.

The checkbox “Long numbers” responsible to textual representation of the current node’s floating point parameters. When unchecked, the only small and hardly fixed number of decimal digits shown, if checked — we show up to 14 decimal digits after decimal point, depending of their values.

The right and the biggest part of DSP node-related controls has display the current values *for selected in the nodes-list node* and has contain uniform controls to change them. We describe them row by row, from up to down. Again, please keep in mind section 3.1 and especially the picture 2. You should remember, that the state (values and accessibility) of this control rows reflect the state and features of *selected* in the DSP-list list box DSP-node.

The first control row of check buttons “Inputs” available for any node at any time. This buttons have responded for connections of the virtual variables to DSP-node mixer. If no input button checked, the node gets zeros as input signal and produced zeros too. For new node there are no inputs selected.

Next control row — output selection radio-buttons. This buttons available at any time for all node types, exclude Master. As you remember, Master node has not output variable. Default output selection for new node — variable ‘Z’.

Next control row labeled “Gain”. It is also available for all DSP-nodes at any time. This row responds for two operations, shown at picture 2: gain adjusts and inversion of spectrum. The numbers in its frame show current gain settings for left and right channels in multiply units and decibels. The check button at the right side with sign ‘=’ has respond for lock / unlock left and right channel setting for this control row. When it has pressed, gain and spectrum inversion controls for right channel are blocked and their state reflect state of left channel controls, when the button ‘=’ free left / right gain and spectrum inversion controls can be managed separately. The gain usually controlled by slider bar, but if you want to set it to exact value, you can press small oblong button w/o label at the right of corresponding bar. It involves small popup text box to enter exact value for corresponding slider bar; press “Enter” to accept the new value or “Esc” to keep the old one. If entered value is not a floating-point number or if it lies out of acceptable range, the control silently keeps its current value. The same buttons with popup edit box to enter exact value are connected to any slider in our control window; if a slider blocked according logic of work, the corresponding button blocked too. At the right from “exact gain” for each channel, we place small square toggle buttons, which sequentially change state of the spectrum inverter. When the buttons label is ‘n’ (normal), I and Q component of corresponding channel stay unchanged, when ‘i' (inverse) — I/Q (real and imaginary parts) are swapped. Note that it is not very useful feature — while the final result is real, its only adds an additional 90-degree phase shift to corresponding channel.

Below gain controls placed “Channel exchange” row. It is also available for all DSP-nodes at any time. The row contains five radio-buttons with obvious purpose — “Normal” keep both channels unchanged; “Swap R/L” swap left and right channels; “Both left” and “Both right” route sound to both channels from left or right channel stream; “Mix down” route to both channels average signal value from both input channels.

Next row of controls manages spectrum frequency shift. It is available only for shift nodes. The row label shows current spectrum shift values in hertz for both channels. In general, the row works like the “Gain” row, but there are some differences. The channel locks toggle button at right side (labeled as ‘^’ at picture 3) while pressing sequentially change lock state in order: … / ‘^’‑pressed / ‘=’‑free / ‘=’‑pressed / ‘^’‑free / …. When the button is up (free) frequency shift can be set for both channels separately. When the button in state “‘^’-pressed”, the frequency shift for channels is mirror-locked, i.e. shift for right channel equal to negative value for the left channel. In state “‘=’‑pressed” shift values for right channel equal to the one for the left channel by the usual way. The toggle buttons with label ‘x’ turn on bypass over spectrum shift modulation stage of DSP-node when pressed; when they free, modulation stage working according setting. The right-channel bypass button locked to the left-channel one while lock button is set to any pressed state.

The last four rows respond for phase modulation. They available only for PM type of DSP-nodes. Their behavior identical to the spectrum shift row, but there is no mirror-lock state to any PM locks buttons. PM stage bypass buttons (toggle buttons with ‘x’ label) stay at the row, which controlled phase modulation frequency. PM control rows reflected the parameters from the formulas (4) and (5) by the next way:

* “Phase modulation frequency” — Fp;
* “Phase modulation phase (int)” — Pint;
* “Phase modulation level” — LPM;
* “Phase modulation angle (ext)” — Pext.

The last control section, which related to DSP-list (in green in the middle-bottom frame at picture 3) is output per-channel I/Q selector. It represented by two pairs of radio-buttons with labels “I(Re)” and “Q(Im)” in boxes “Master Left” and “Master Right”. This buttons always connected to the Master DSP-node, regardless selection of list box with DSP-list and any channels locks. They determine which component of complex-valued DSP-list output signal will be routed to real-valued plugin output. There is good idea to leave the both in “In-phase (Real)” position; if you switch to “Quadrature (Imaginary)” position — you obviously add 90-degree phase shift to corresponding output signal.

Now you might be ready to make small example — DSP-list from two nodes — Master and Shift, which will play the sound with summary spectrum shift to 4 Hz — the left channel will be shifted down to 2 Hz, and the right channel will be shifted up to 2 Hz.

Start WinAmp. Add a CWAVE or WAV file (if you set support to the one). Select this file in playlist window. Press Alt+3 — our control window might be to appear. The DSP-list for now must contain only one node — Master, if not — press “Del All”.

Setup the Master. In “Inputs” check variable ‘A’ (we will route to it output of Shift node); make sure, that the other entire Master’s input variables, include ‘In’ unchecked. Set the gain locks to “locked” (‘=’ pressed) and set by the left slider output level to 0.6. Check, that the spectrum inversion set to normal (‘n’) and channels exchange set to “Normal” too.

Press “Add…” to create Shift node. In dialog from Picture 4, type a name (“-2/+2=4Hz”, for example), select type “Shift”, press “Create”; the new node in DSP list will appear — its name will be “Shift--2/+2=4Hz”). Check, that this node selected in the list box.

Let’s make setup for new node. Select ‘In’ from input variables (other shall be unchecked), select ‘A’ as output; locked gains set to 1.0, no spectrum inverse / no channels exchange. Check the frequency shift mirror-locked (‘^’ pressed and corresponding right channel controls has disabled). The default shifts for new shift node is +2.0/-2.0 Hz. Change it to -2.0/+2.0 Hz by corresponding slider of the left channel.

Play the track. If all of the changes has made in right way, you will hear the sound with selected shifts. Close our control window. Press right mouse button at the selected file. In context menu select “Send to”/”Format converter”. Never uncheck the “Show this before each convert”! Set appropriate path and name for the output file. Then in “Encoding Format” select “WAV Encoder v1.02a. In the “Wav Encoder Options” the box “Write RIFF header” should always be checked and “Convert to format” — unchecked. Extension, of course, should be “WAV”. Press OK.

If you don’t touch the other setting, you must get WAV-file with selected (24- or 16-bit) resolution with “shifted” signal.

So, now we are learned all about the most complicated part of plugin. Let’s complete the rest.

#### 3.2.3 Sound rendering control

*For many years while we working with complex sound, we do not take a care about dithering and noise shaping. It was wrong! This technic completely changes the sound imagination vs. simple rounding to integers! Moreover, while implementing we met some “Black Holes” in subject; especially in the noise shaping area. Particularly, we found (but not with very clean understanding for the moment), that while the world of complex sound transformation significantly different from usual real-valued, the usual dithering / noise shaping approaches may be not optimal in the case. So, please carefully read about and do not hesitate, at least, to increase amount of dithering noise. Remember, that for many of ten years people get pleasure from the recorded music sound with signal / noise ratio less than 30 or even 20 dB…*

The setting of resolution of output sound (24- or 16-bits) considered in section 3.2.1. Here we describe other controls, which related to conversion of double valued DSP-list output to 16- or 24-bit integers. If to go by the order at picture 3 from left to right, the first control, named “Render type” select the type of dithering. This drop-down list provides five variants:

1. Simple round to integer — no any additional noise added;
2. Rectangle PDF dithering — dithering with noise with rectangular probability distribution function;
3. Triangle PDF dithering;
4. Sloped TPDF dithering — the same as TPDF, but each noise sample make as sum of the current rectangle PDF value and negated previous one (plain TPDF used two *independent* RPDF values);
5. Gaussian PDF dithering — we use sum of 12 RPDF-distributed independent values to approximate the Gaussian distribution — to avoid low-probability extremely high values.

In general, we recommend using TPDF or “Sloped TPDF”. Please note, that Sloped TPDF mean some filtration of dithering noise with the almost same result the noise shaping intends to, i.e. the noise keep all of the statistical characteristics for dither effect, but became much quiet subjectively. In many cases, it has allowed to avoid taking a deal with the Black Magic of noise shaping.

Next control — toggle button “Quantize”. It selects the quantize type — “Mid Riser” type or “Mid Tread” one. The “Mid Riser” type mean (in our case, because integers in complementary code have 1-wider diapason for negative numbers then to positive) that all of the input numbers in (‑1, 0] give output integer -1, numbers in (0, 1] give 0, etc. The “Mid Tread” quantize have amplitude characteristic has right shift to 0.5 from previous, i.e. input in (‑1.5, ‑0.5] give -1, (-0.5, 0.5] give 0, etc. For the first touch its look more adequately (symmetrical over zero), but we have recommended “Mid Riser”. Really, there is no clean agreement about; the resulting difference between both approaches is *very* few.

The next *two* controls — “Bits to dither” — the text field to type (non-negative float from 0.0 to 23.0) and the button ‘<’ to accept the value; the number typed will not be accepted until you press the button. The value set maximum amplitude for TPDF / Sloped TPDF dithering noise, i.e. if you set it to 1.0, TPDF/STPDF dither numbers will have values in (‑1.0, 1.0) interval. Other distributions amplitude will be reduced to the same dispersion (and root-mean-square — RMS) value than *TPDF with the designated amplitude*. In other words, the loudness of rectangular and Gaussian noise for the fixed “Bits to Dither” will be exactly same as loudness of triangle-distributed noise for the value. Note also, that subjectively “loudness” of Sloped TPDF will be less, but if you have make a corresponding measure it in your favorite sound editor, you got identical results. What do you want? This is Black Magic of differences between the “distribution” and “spectrum” terms.

Note that many sound editors, which have gave to user an ability to choice of PDF for dithering, have used another approach to select noise amplitude for rectangular and Gaussian PDF; but we make our decision.

The next control in the group labeled “Noise shaping”. It provides 18 variants; all of them was taken from SoX sound editor repertoire:

1. Flat (no noise shaping);
2. [44K] F-Weighted, FIR;
3. [44K-48K] Modified E-weighted, FIR;
4. [44K-48K] Improved E-weighted, FIR;
5. [44K] Lipshitz, FIR;
6. [48K] Shibata, FIR;
7. [44K] Shibata, FIR;
8. [38K] Shibata, FIR;
9. [32K] Shibata, FIR;
10. [22K] Shibata, FIR;
11. [16K] Shibata, FIR;
12. [11K] Shibata; FIR;
13. [8K] Shibata, FIR;
14. [48K] Low Shibata, FIR;
15. [44K] Low Shibata, FIR;
16. [44K] High Shibata, FIR;
17. [44K] Gesemann, IIR;
18. [48K] Gesemann, IIR.

Please note that any noise shaping filter intend for particular sampling frequency. We can say about right choice very few. First of all, avoid using nose shaping, if you allow at least a few signal clipping, because standard “linear” noise shaping (which we implemented) can catastrophically increase the quantize saturation effects! Moreover, we are almost sure, that standard noise-shaping loop is unstable by design, but it is a subject for some different research paper. In this context, we can to say (if to consider variants 1–3), that F-Weighted and Improved E-weighted noise shaping produce more pleasure noise, but the noise shaping loop much more stable with Modified E-weighted filter. Gesemann filters also produce nice sound. We also do recommend avoiding big amount of noise together with noise shaping — in some random point the noise-shaping loop can fall into generation mode even without any signal saturation and completely kill the sound.

The last small set of controls in the group — clips counters, separate for left and right channels. They show in real-time how many samples were clipped by both playback and transcode plugin interfaces. The small square button at right side from the counters with label ‘x’ instantly reset the counters to zero (this do not an affect to the sound). The counters are helpful to set the right signal level; you should to avoid getting the clips. Note that any (digitally?) compressed (especially highly over-compressed — like most of the modern pop-music recordings) sound after filtering by *any* implementation of Hilbert transform *significantly increase its amplitude values* while keeping the RMS (up to 6 dB or even more) — we are sure, that this is very interesting from *many* sides phenomenon. We call it “***Hilbert’s fluffy***”.

The switch button “Excepts” control monitoring of floating point exception. We describe it latter. It intent for some numerical experiments which does not have immediate relationship with plugin functionality; we describe it latter.

#### 3.2.4 Embedded analytic signal (“Hilbert’s”) converter

In this (last in our list) group of controls most significant is the drop-down list “Embedded Hilbert’s transformer type”. It related only to WAV (“RWAVE”) files, and do not have any effect for work with CWAVE files. We provide three pre-calculated Half Band Low Pass IIR filters for our transformer:

* HB LPF Type 0 (weak, order 15);
* HB LPF Type 1 (strong, order 19);
* HB LPF Type 2 (medium, order 18)

In most cases, you should use Type 1 (default). As we already wrote, the embedded analytic converter is a complicated pass-band filter. To say roughly, its rejection band (both in lowest and highest frequencies, but for our applications only the lowest are critical) determine by transition width of HB LPF, which it used (for further details, see references in the section 5 “Our Thanks”). When we make Type 1 filter, we have want to solve the problem — to keep as many low frequencies as possible. On this way, we make some trick — this filter is not pure half band, its transition band is slightly shifted. We got the transformers pass band, which begin from 20 Hz, but this approach adds some (small) distortions at lowest frequencies. The type 0 and type 2 filters is pure half band. Type 0 theoretically provide minimum own distortions, but with it the frequencies below 110 Hz will be rejected (all of the frequencies have related to Fs=44100 Hz). The Type 2 filters, as usual, something medium. You can find the complete design parameters sets for all of the filters in the source file hblpf.c.

The frame “Sample counters” contains two identical groups from tree controls each for both playback and transcode plugin interfaces. (Please note, that XMPlay do not use plugins transcode interface at all, and used playback interface for transcode the files.) Each group contain sample counter (more precision, the value (n mod Fs) from (2) and (3) formulas). At right side from sample counter placed the button with label ‘x’, which reset to zero current value of sample counter. If you press it during plugin worked, it makes some click. Another button, labeled “X Hilb” reset corresponding analytic signal transformer; it has not any affect for handling CWAVE files, and produced click while plugin working with WAV files.

In general, you do not need to care about these controls. The activity of sample counters may be useful to see plugin activity, especially, while you tune the XMPlay to handle WAV-files via in\_cwave — if the file played or transcoded, but the sample counter stays unchanged — something goes wrong. The possibility to reset the counters / transformers may be slightly useful for resetting plugin to “fresh-start-state” before handling some group of the files.

### 3.3 Plugin’s Config file

Plugin’s config file has name “in\_cwave.cfg”. It is keep all of the setting from main control window (exclude “Show this by ALT+3” value) and all of the setting of the system setup (“Bugs management”) window. It lives in subfolder “in\_cwave\” somewhere in Windows current user’s application data, in our Windows10 system the full path look as:

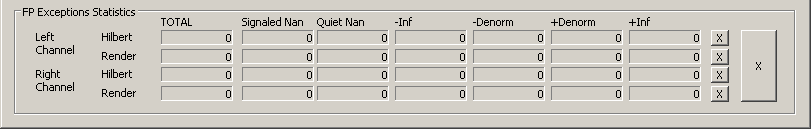
c:\Users\username\AppData\Roaming\in\_cwave\in\_cwave.cfg

In other Windows systems, the path can be slightly different. The config is a text file, which encoded in UTF16-LE with Byte Order Mark (BOM). Note, that it is not intend to manual editing — you cannot add to it or change in it something, which you cannot set via corresponding GUI (exclude, probably, DSP-list name prefixes). Just in case note, that all of the double values kept in config file in form of binary (hexadecimal) images — like unsigned 64-bits integers in format “0xHHHHhhhhHHHHhhhh”, so the double “0.8” kept as “0x3FE999999999999A”. In principal, config should understand “normal” doubles — if you change corresponding hexadecimal in C-notation to double in C-notation, it might work. However, while you save it, all of the doubles will be represented by their binary images again.

We highly not recommend changing something in this file manually. Our config.c source is a child of tired evening — it just works with self-written files and wants to be replaced to something like XML / JSON or so. However, there is out of the plugin functionality while it written in pure C-language.

### 3.4 Floating point exceptions monitoring

If take a look to open source audio programs with floating point data path, its ease to note, that many authors take some care about some handling of special floating values during DSP-computation — like NaN, Inf or so. Theoretically, there are two places in our plugin, where such values can be produced — the (high-Q) IIR filters in WAV data to analytic signal converter, and, of course, noise shaper. To have ability to catch the exceptional situations, we made special implementations for corresponded code, which allows counting floating-point exceptions by corresponding potentially dangerous DSP algorithms. To activate this feature, check the button “Excepts” in the group “Clips” in main setup window (See Picture 3). The next table will appear at the window bottom:



Picture 5. The floating-point exception counters.

To return to normal view and normal computation, uncheck the button “Excepts”. Note, that while the button “Excepts” unchecked, all special values counters stay zeroed and do not count. The four small buttons with “x” at Picture 5 resets the corresponding counters; the big button reset all counters. Please note, that while exceptions counting is active, the plugin consume ***much*** more CPU resources, than in normal mode — probably up to playback interrupts on some slow old systems, so, if have no any ideas how to use the counters — simple not check the button.

## 4 About CWAVE files

As we explained, while in\_cwave is CWAVE-file input plugin, it can work without CWAVE files. We have some converters, which works, probably, slightly better, that embedded one, but the code is out of publication quality, and we have not a time to refactor it. Probably we make it latter, probably not. However, you can make some by yourself — it is rather easy.

First, look at our cwave.h header. It describes format of the file. The file contains a fixed size binary header and uncompressed array of binary complex-valued samples. The samples can be packed in four forms. For most of the cases the form float32 (real part) + float32 (imaginary part) is good. Note that while your analytic converter contains pure Hilbert transformer, i.e. keep the original sound as real part of analytic signal, the format int16 (real part) + float32 (imaginary part) will be interested. Note also, that all floats in CWAVE files are normalized to [‑32768, 32767] interval like converted to floats / doubles 16-bits signed integers.

Some words about the header. Cwave.h declares two versions of the header — V1 and V2. The difference only that V2 contain CRC–32 of the sample data in the field, which reserved as alignment placeholder in the V1 header. The exact algorithm for CRC–32 computation contains in the files crc32.h / crc32.c — it has given the same results as CRC–32 in zip-files utilities. The mystic parameters k\_M and k\_beta related to times when we have a deal with naive approach of classical Hilbert FIR with Kaiser window function — in this fields we kept filter order and Kaiser window β-parameter. Now we usually set k\_beta to zero and k\_M to ‑1 for DFT-based and ‑2 to other algorithms. In general, these parameters do not affect to something, and you can keep in it arbitrary numbers. You can see their values in our file info window, and it is all.

Some notes about conversion to analytic signal algorithms. First — do not waste a time to faze splitters![[1]](#footnote-1) The best which we have seen, called “hiir” (“An oversampling and Hilbert transform library in C++” by Laurent de Soras, 2005-2013, <http://ldesoras.free.fr>) — uses polyphase elliptic filter design approach. As we discovered, its produce lots of phase garbage at low and mid-low frequencies, even for rather high orders; especially for real sound in contrast to embedded (short linear-frequency modulated signal) test; we are sorry. From the other side, as we have saw in the hiir’s filter designer code, it uses old *simplified* algorithm to compute elliptic filters coefficients, but we disbelieve for a chance to improve this approach significantly. The faze splitter always make phase mistakes at the edges of band and this looks for us as this approach incompatible with any audio applications (V2.0.0 @ 2016). However, here we, probably, wrong (V2.2.2 @2020).

Another popular approach — naive Hilbert FIR-filter. There are a lots of open source implementations for it with different window functions from different textbooks; the SoX Audio Editor for example (author Chris Bagwell; Hilbert filter implementation by Ulrich Klauer; see [sox.sourceforge.net](http://sox.sourceforge.net) for detail). This filter provides ideal phase shift, but unfortunately has irregular amplitude-frequency characteristics, especially at the edges of signal band but in all band too. This has produces very unpleasant affect, which, in frequency shift meaning, equivalent frequency-depended channels cross-talk. In addition, of course, the 32K-limit to FIR-filters order, hardcoded in SoX FFT-based FIR-kernel looks insufficient even for the sampling frequency 44100 Hz. Nevertheless, there is one ***very interesting thing*** related to Hilbert’s FIR. The Adobe Audition from the version 3.0 has VST-plugin, which called “Graphic Phase Shifter” (GraphicPhase.dll, for Audition 3.0 it could be found at [ftp.adobe.com](ftp://ftp.adobe.com) as Audition support files in archive “audition\_3\_0\_graphic\_phase\_shifter\_update.zip”; as we have seen, it has included in more recent version of Adobe Audition distribution out of box. (Do not miss it with an older Audition .XFM (non-VST) phase shift effect.)). This plugin provides arbitrary frequency depended phase shift for input signal, and, as particular case, can make pure Hilbert transform. As any can to see, this plugin has calculated some sort of *complex* convolution via FFT, i.e. implement some complex FIR-filter. However, while it makes Hilbert transform with FFT size 64K (hardcoded maximum), we can see, that it produces (almost?) ideal 90-degree shift together with very good amplitude-frequency characteristic, this looks absolutely unreachable for “usual” naive Hilbert’s one. Probably it’s the best Hilbert transformer in the universe, but for the moment we misunderstand how it works. However, we do not test it hard.

In addition, of course, our current workhorse — “dumb” DFT-based analytic transformer has rather sufficient results. We know about all disadvantages of this approach — the Gibbs phenomenon particularly, but have found that the result for long enough sound fragments is acceptable. Note, that this approach has additional abilities — to clear unwanted lowest and highest frequency components, for example.

There are other interesting things in the world, Mathsoft’s Matlab IIR Hilbert transformer, for example. But quick look for it show us, that from one side, the theory about looks complicate (what do you know about Hankel’s matrixes?). Really, as we discovered, it works not better then chosen for in\_cwave approach taken from Serge Bahurin’s analytic transformer (www.dsplib.ru / www.dsplib.org, see Our Thanks section), which has based on clear and beauty math together with clean and beauty physical meaning.

So, the sufficient analytic transformer for audio signal — interesting and complicated task and we do not want to deprive you from the pleasure to find the best solution for yourself.

## 5 Our Thanks

We, Rat and Catcher Technologies, in deep appreciation, has to say lots of thanks to:

**David Johnston**, former www.syntrillium.com, the CoolEdit (now Adobe Audition) root author — his works open the subject for us.

**Serge Bahurin**, www.dsplib.ru / www.dsplib.org — we take many ideas from him. Especially — for his non-trivial analytic signal transformer, which we implemented in our plugin. In addition, we used his dsp.dll library for calculation elliptic HB LPF filters coefficients (unfortunately, this binary now is unavailable — it was removed from the site by the author as “unsupported”). Many thanks for him.

**David Bryant**, [www.wavpack.com](http://www.wavpack.com), author of WavPack — his source of WinAmp’s plugin in\_wv for WavPack playback is a single point, where we found information about WinAmp transcoding API.

**Chris Bagwell et al**, [sox.sourceforge.net](http://sox.sourceforge.net) — from the SoX sources we learn about sloped TPDF dithering, as well as we have taken some noise shaping filters coefficients sets from their collection. The noise shaping empirical curves from their pages on Sourceforge let us to check our noise shaping implementation.

**Makoto Matsumoto and Takuji Nishimura** — for their wonderful Mersenne Twister pseudorandom number generator (now is the part of C++ STL).

**WinAmp and XMPlay players’ authors** — for high usability sound players. Their open, simple and flexible API make the programs much more, then computer sound players.

1. By term “faze splitter”, we mean two all-pass filters designed to make (near) 90-degree shifted outputs. In some wider meaning, the approach, which implemented in our in\_cwave, is also “faze splitter”, but here we talk only about linear all-passes. [↑](#footnote-ref-1)