# Cool Edit

Version 1.51

User’s Manual

Syntrillium
SOFTWARE CORPORATION

P.O. Box 60274 • Phoenix, AZ 85082-0274
602-941-4327 • fax 602-941-8170
syntrill@aol.com

# Cool Edit

*Version 1.51*

***Cool Edit is a digital sound editor for Windows. You might think of it as a paint program for audio-- just as a paint program enables you to create images with colors, brush strokes, and a variety of special effects, Cool Edit enables you to "paint" with sound: tones, pieces of songs and voices and miscellaneous noises, sine waves and sawtooth waves, pink noise, white noise, and brown noise-- even silence. Cool Edit also gives you a wide variety of special effects to "touch up" your sounds: reverberation, noise reduction, echo and delay, flanging, filtering, and many others. Cool Edit is both flexible and powerful. It works with dozens of audio file formats, supports files up to 1 gigabyte in size, enables you to customize any function to achieve just the effect you want and then to store your settings as a "preset", and offers scripting and batch-processing features so that you can automate your sound-processing routines.***

***This manual describes all the features and procedures you need to make the most of Cool Edit. Once you've started working with it, you'll understand why thousands of musicians, multimedia authors and artists, scientists, telephony professionals, and audio enthusiasts around the world have chosen Cool Edit as the ultimate sound editor.***

***But first, a few words:***

We hope you find Cool Edit a joy to use! Please see the section on Configurations if you have any problems running the program. This may be the first application you have used that takes full advantage of your PC (lots of hard disk space for temporary storage, lots of floating point math for many of the waveform editing functions, lots of data bandwidth for recording and playing - especially 44.1K 16-bit stereo CD quality audio, and lots of CPU time while crunching out the results of various waveform functions). In many ways, it is similar to a CAD package, except that the focus is on audio instead of graphic design.

Please use this documentation for reference purposes, or just to learn more about digital signal processing and all the capabilities Cool Edit already has. We are committed to keeping Cool Edit up to date with current and future technologies.

David Johnston
Robert Ellison
Syntrillium Software Corporation

# Contents

## Features Overview

- **Multiple instances** (copies of Cool Edit) can be loaded, copied to, and pasted from in any mixture of wave formats.
- Cool Edit can synthesize just about any sound using the **noise** and **tone** generation functions along with the various **wave transformation functions.**
- Convert between various sample types with variable levels of quality for upsampling and downsampling.
- Any sample can be used as an "instrument" and set to **music**, like a dog barking Jingle Bells, or Tarzan yelling the national anthem.
- Waveforms of any size can be edited, limited only by hard drive space
- Supports PCM, Microsoft ADPCM and IMA ADPCM .WAV, Sound Blaster .VOC, raw PCM, Apple AIFF, and ASCII Text **data formats.** MPEG Layer 2 file format available upon request.
- Add 1960's sounding special effects using the **Flange** function.
- Continuous echo of all or part of a sample with filtering for successive echoes possible with **Echo**.
- Simulate the acoustics of any environment with the **3D Echo Chamber**, and **Reverb** functions.
- Speed up or slow down samples without affecting pitch, or raise and lower pitch without affecting tempo by using **Stretch**.
- Distort your waveforms to give effects such as low quantization, blown car speaker, fuzzy cheap amplifier, and overdriven tube amp.
- Supports descriptive **information**, and **bitmaps** when saved in .WAV format.
- Supports a **cue list**, and a **play list,** when saved in .WAV format for playing portions of a wave in any order, with looping.
- Built in **CD player** when the [MCI] CD Audio driver is loaded, and a valid music CD is in the drive.
- **Brainwave synchronization** function available to create sound files that alter your state of mind.
- Record and Playback waves using any Windows compatible sound board, even works with the PC Speaker driver.
- Spatially locate sound sources to appear as if they are coming from different directions by **delaying** one channel a few microseconds in stereo waveforms.
- Most functions have programmable **Presets** to easily save and retrieve your favorite effects.
- You can feel free to experiment, and easily back up a step if you don't like what you did by using the **Undo** function.
- 8-band **Quick Filter** for quickly adjusting equalization.
- **FFT filtering** and **Spectral View** make it easy to filter and analyze audio with high precision.
- Compress/Expand/Limit dynamic range using the **Compressor** function. Can also act as a noise gate to silence audio between spoken words.
- **Noise Reduction** possible to reduce any type of noise up to 20dB, including broad band noise and that annoying 60 cycle hum.
- Practically any file format can be loaded successfully into Cool by using the raw PCM file type, and then **Adjusting** the sample rate, bit rate, and number of channels, as well as performing **Byte Swaps** to get the byte ordering and sign interpretation correct.
- Skewed waveforms can be centered properly using the DC Bias filter, and waves can be normalized, faded in, faded out, or panned by using the **Amplify** and **Normalize** functions.
- **Frequency analysis** can be continually be performed on the waveform at the cursor to better see exactly what frequencies are present.
- Support for unlimited wave formats by allowing others to write custom file formats that Cool Edit will support. See the **File Filter API**.
- **Cool Scripts** will remember everything you did to arrive at a certain waveform, or transformation and remember it for the future. A **Batch** feature lets you run a script on a group of files and save them back in any format.
- Customizable **Toolbar** (with a little work) possible by modifying the cool.ini file.

- Assign keyboard keys to play specific parts of your waveform by using the **cue list**.
- Ring Modulation, or any type of modulation possible via **Generate Tones** or **Paste Special**.
- Ultimate distortion effects possible with the **Distortion** function.
- Generate telephone tones using the **DTMF Tone Generator**.

## How Cool Edit Works

### Overall Design

Cool Edit currently uses the Single Document Interface (SDI) design paradigm. This means that an instance of the program operates on a single continuous waveform. This, however, in no way means you have to work on only one waveform at a time. Multiple copies can be run at once by choosing File:New Instance. Use multiple copies to temporarily store waveform snippets (like having a multiple visual clipboards) or to work with more than a single waveform at a time. Since each running copy of the program is totally independent of the others, you can even play waveforms and work on one copy while the other is busy calculating.

While Cool Edit is calculating, most of your CPU's time will be used for these calculations, although you can still run other Windows applications albiet a little slower. If in the middle of calculating a lengthy operation you need the processor for another task, you can suspend, or pause calculation by pressing the Pause button in the progress meter dialog (or just clicking on the blue progress meter in some versions).

In general, all functions that alter existing waveform data can be found in the Transform menu, while those that create new waveform data can be found in the Generate menu. All file and main program control functions are in the File menu. All functions dealing with the general purpose editing can be found in the Edit menu. And finally, all miscelleneous functions that did not fit anywhere else are in the Options menu.

As often as possible, I have included help [?] buttons for quick help about the current dialog. Lengthy explanations of all the functions can be looked up this way.

### Temporary Files

When Cool Edit loads a file, it is actually creating a temporary *backup* copy of the wave, uncompressing it if necessary (which is why the size displayed in file size window can be larger than the size of the file that was loaded). All editing, transforming, generating, recording, and playing is done by using this temporary file. Most all systems are fast enough to record and play directly from disk, so the limits of RAM are removed allowing for the editing of files as large as your hard drive. Cool Edit uses the temporary directory specified in the **Settings** dialog. (This dialog first gets the directory from your TEMP environment variable, usually set in your AUTOEXEC.BAT file, the first time Cool Edit is run.) The temporary directory used to store Undo information can be different than the main temporary files directory. Undo and various other functions create temporary files that are all removed when every instance of Cool Edit is closed.

### Playing and Recording Audio

Cool Edit uses a multi-buffered system for recording and playing audio. The total buffer size can be specified in the Options:Settings dialog. The size is measured in seconds, and should be just slightly longer than the longest time you expect your system to be *busy*. The system is busy whenever you see the hourglass cursor, so this should give you some indication of the proper time to enter. If the system is busy for longer than the buffer time specified, recording or playing will stop. You will hear gaps of silence during play, and missing audio data from recording if this value is too low. For most systems, 4 seconds is fine, and 8 seconds should be more than enough.

## Configurations

Cool Edit doesn't need much configuration, but for optimal performance, there are a few guidelines you can follow.

### Location of Temporary Files Directory

Your temporary files directory should be located on your largest, fastest hard drive (of course :-). A slow hard drive may not be fast enough to record to at higher data rates (like 44.1K, 16 bit, stereo). Also, compressed (or stacked) hard drives require more processor time, and may make recording at higher data rates impossible on slower systems because of the time needed to encode, or compress the data. You want the hard drive with the greatest amount of free space on it as well, since this will limit the size of the file Cool Edit can work with. Be aware that if the wave you are currently working on takes up more than half of the available space on the temporary drive, then you will not be able to save that wave to that temporary drive in a non-compressed format, since the saved file will take up more room than the drive has left. The temporary files directory can be specified in the **Settings** dialog.

Internally, this setting is kept in the TEMPOVERRIDE= line in the [Size] section of COOL.INI. The file COOL.INI is located in your Windows directory. After the section that reads "[Size]" you may enter the line, "TempOverride=D:\TMP" for example.

### Interrupts and DMA Channels

Many sound boards use Interrupts and/or DMA channels. For 16-bit audio, you must specify a DMA channel of 5 or greater. Consult your sound board user's guide for more information. For Cool Edit to run at its fullest, these must be set correctly. You can still edit waveforms of any sample type even if your board is not capable of playing them, and even if you do not own a sound card!

Some problems occur during recording or playback if the Interrupt or DMA settings for your card conflict with some other installed device (hard drive, network, etc.). Some devices, like SCSI hard drives, may steal too much of the data bus away from the sound card, not allowing it to run at higher data rates. So if you have any problems playing or recording, especially at higher rates (44.1K and/or 16-bit) please check all the settings on all your hardware to ensure there are no conflicts. Cool Edit plays audio directly from the hard drive, so your computer's data bus must be able to handle the traffic coming from the hard drive to memory, and from memory to your sound card at the sample rates you wish to use the program at.

**Navigating About the Interface**

The **Toolbar** represents the commonly used functions as icons along the top of the window. A quick help description of the button's purpose will pop up if the mouse remains over the button for about a half second. If too annoying, this feature can be turned off in the Settings dialog.

**Waveforms** are displayed in green.

- Any portion may be selected by clicking on the waveform with the left button, and dragging left or right.
- Holding the **SHIFT** key while doing so, or using the right mouse button will increase or decrease the size of the currently selected portion.
- **Double-Clicking** on the waveform will select the entire visible wave.

The**View Indicator** (the green bar above the waveform) depicts which portion of the entire waveform is being viewed in the workspace below.

- **Sliding** the green bar (when present) left or right scrolls various parts of the waveform below into view accordingly.
- **Clicking** to the left or right of the green bar will scroll the audio to the left or right one screenful when zoomed.
- **Double-Clicking** on the View Indicator will bring up the Viewing Range dialog to allow direct entry of starting and ending samples for the portion of the wave to view.

The **Yellow Arrows** indicate the point of insertion. Clicking anywhere in the workspace will move the point of insertion to the mouse.
The **Red Arrows**, when present, represent the Cue List entries for cue markers.
The **Blue Brackets**, when present above and below the waveform represent the Cue List entries for a range.

Choose **Zoom In** to expand the selected portion to the full window. **Zoom Out** will give a larger view of the waveform, while **Full View** will display the entire waveform in the workspace.

Playing and recording is controlled by the lower row of buttons.

- The **Play** button will play the portion of the wave that is currently being viewed, or the portion that is highlighted.
- The **Pause** button will temporarily pause the playback or recording of audio. The button turns into a **Continue** button when audio is paused. If recording, the red record bar turns yellow to indicate a paused state.
- The **Record** button will start recording at the current insertion point. Any waveform data after that will be recorded over.
- Use the **Stop** button to end waveform playback or recording.

The **Record Level Meter** below the Play/Record buttons will display red when a waveform is being recorded, and a real-time display of the input when Monitor Source is chosen. The display turns yellow if recording has been temporarily paused.

- **Double-Clicking** on the Record Level Meter will start or stop monitoring.

The **Time Display** windows show the current Starting and Ending points of the current selection, or portion being viewed.

- **Double-Clicking** on these windows will toggle the display between time in seconds, and in samples.

The **Wave Format** window displays the format of the wave in Channels, Sample Rate, and Bits Per Sample.

- **Double-Clicking** on this window will bring up the Adjust Sample Rate dialog to change the way Cool Edit interprets samples.

A **Play Position Indicator** (vertical inverse bar) shows the current playing position when a wave is played.

**The Cool Edit Paradigm**

The main purpose of Cool Edit is act as a *non real-time* studio to edit digital waveforms. Many of the functions take longer than real-time to process, thus there is no non-destructive "playthrough" of any of the effects. So don't go throwing away your digital effects processors just yet.

Only one waveform can be edited at a time, but you are not limited to the number of copies of Cool Edit that can be running. Running multiple instances does not use very much memory, so don't be afraid to open as many copies of Cool Edit as needed. Instances of Cool Edit can be worked on while others are in the middle of processing audio, since each copy is completely independent of the other. Yet, copying and pasting can be performed seamlessly between each of the instances.

When a file is opened, a temporary backup is made, and all edits are made on the temporary backup copy. Changes are only made to the original file if they are explicitely saved back. Choosing File:Re-Open will re-copy the original waveform to the temporary working copy at any time (kind of like a global level Undo). If you are using Cool Edit to just play a waveform, and not edit it, it is better to use some other wave player program, since Cool Edit uses hard drive space to create a backup copy before it can be played.

The waveform currently being edited is always resident on the hard drive (except for during the middle of operations). Because of this, waveform sizes are limited only to the amount of free space on the temporary drive. The maximum size Cool Edit allows is just over 1 gigabyte, which should be plenty for most purposes. Because the temporary file takes up hard drive space while Cool Edit is open, you have less space on the temporary hard drive to save your files. Editing a file of more than twice the original free space on the temporary drive will make it impossible to save the waveform on the same drive, unless a compression method is used (eg ADPCM).

Waveforms are played directly off of the hard drive as well. For this reason, it is important for the temporary hard drive to be as fast as possible. To improve hard drive speed, remove any type of disk compression you may be using if your working on a slow processor system and desire to record at high sample rates. But, waveforms of any size can be played.

**Keyboard Shortcuts**

| | | | |
|---|---|---|---|
| Delete | Delete | ALT+P | Play waveform |
| CTRL+C | Copy | ALT+S | Stop play |
| CTRL+Insert | Copy | SPACE | Play waveform / Stop playing waveform toggle |
| CTRL+X | Cut | F2 | Repeat last command (show dialog if command has one) |
| SHIFT+Delete | Cut | F3 | Repeat last command ***now*** (using last used settings) |
| CTRL+V | Paste | F8 | Add the current cursor location or selection to the cue list |
| SHIFT+Insert | Paste | ALT+M | Monitor Source signal toggle on/off |
| CTRL+T | Trim | ALT+I | Waveform Info |
| ALT+Bksp | Undo | ALT+Z | Frequency Analysis Window |
| CTRL+A | Select Entire Wave | | |
| CTRL+S | Select the current view | | |

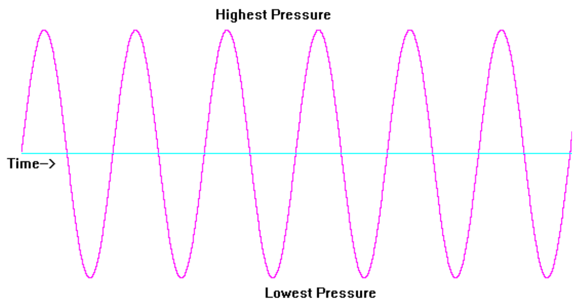
**Mouse Shortcuts**

Left Click and Drag on waveform to highlight and select a range of samples
Right Click (and drag) on waveform to extend selection
Shift+Left Click (and drag) on waveform also to extend selection
Double-Click on view indicator (green bar) to enter viewing range directly in samples
Click to the left or right of green bar to scroll view left or right one screen
Double-Click on Levels Meter (black bar) to start/stop monitoring
Double-Click on Sample type display to change sample type interpretation
Double-Click on time windows to change time format
Double-Click on title bar to Maximize/Restore
Rest mouse over toolbar button to get explanation of button's function

## A Short Course in Digital Signal Processing

### Signals (or Waves)

Waves in the context of Cool Edit are sound waves. A sound wave can be written as the how the air pressure on your ear changes over time. When you hear a loud sound, the pressure on your eardrum is greater, and it vibrates harder. Soft sounds affect the eardrum very little, and thus are noticed as softer sounds. So a wave is a convenient representation of how the sound level varies over a time interval. The illustration is a sine wave of a constant pitch. It shows the sound pressure oscillating from low pressure to high pressure and back. In natural systems, this motion follows the path of a sine wave when graphed. The wave here is of a constant frequency and constant amplitude. Choose any time along the Time axis going from left to right, draw a vertical line up and down, and there will be exactly one spot where the wave crosses this vertical line. This is because a wave can have only one value at any instant in time. For example, there can not simultaneously be two different pressure levels on the eardrum at once. If two sounds are heard at the same time, the pressure levels from both of them are simply added, and a single resultant pressure is observed (and at that instant, it is impossible to tell exactly which role each sound played in creating this single value). So a waveform is depicted as a line that can vary up and down freely, going from left to right, but with no "backtracking" (e.g. a graph of a waveform will never look like a circle, or a "U" on it's side).

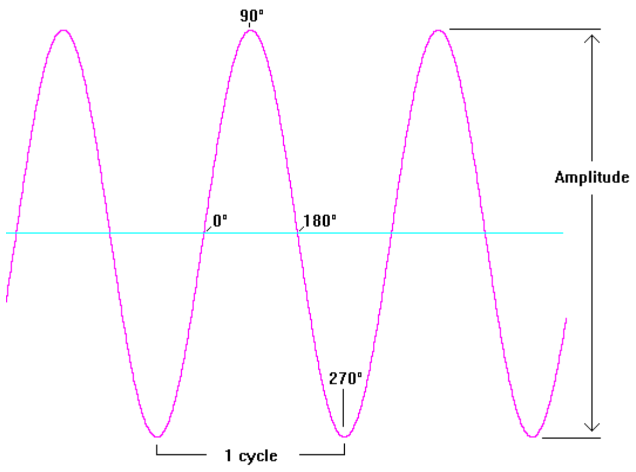


Waves in the natural world are continuous, which means that no matter how much you "zoom in" to the waveform, or no matter how small of a time interval you look at, there are an infinite number of values needed to represent the progression of the waveform during that interval. Other types of waves exist besides just sound waves. Seismic activity can also be viewed as a wave - as in the shock wave during an earthquake. The Richter scale graphs (the familiar graphs of earthquake activity they show on television after a quake) are a prime example. There is a single needle that sways back and forth leaving a mark on a slowly turning cylinder. As the seismic receptors placed into the Earth pick up vibrations, the electrical impulses are sent to the device, causing the needle to sway in response to the movement of the earth.

Cool Edit's normal Waveform View displays waveforms just as described, as a plot with the time going from left to right, and at each instant in time there is exactly one value for the waveform's instantaneous amplitude, or pressure level.

### Amplitude

The amplitude of a sine wave is the difference between the highest part of the wave and the lowest part. The difference between the high and low pressure parts. A low amplitude, quiet wave would be one that would vary much less up and down, while a louder waveform would vary much up and down. Amplitude is generally measured in decibels, although the decibel (dB) itself not an absolute measurement like Fahrenheit is for temperature, but instead is a measurement of ratio. If one decibel is the quietest sound someone can hear, then the loudest sound one can hear without damaging the ears with prolonged exposure would be about 100 dB. Normal speaking would be at about 20 dB. Ten decibels is an increase in volume of 10 times. The decibel scale is not linear, but logarithmic, which means that 20 dB is not 20 times louder, but instead 100 times louder (10 times louder than a 10 dB increase).



### Frequency

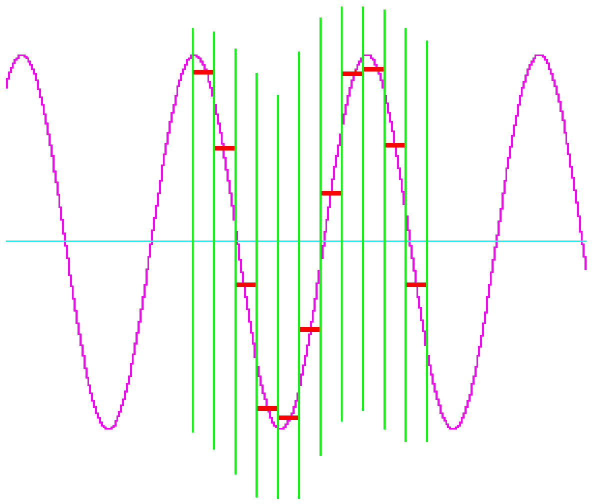
The frequency of a wave determines the pitch we perceive, and is measured in cycles per second, or Hertz (Hz). As seen on the graph, the time it takes the wave to complete one cycle is the time it takes to go all the way from the point of lowest pressure, on up to highest pressure, and then back to lowest pressure where it started. A cycle can start anywhere, not just at the bottom of the wave. The cycle will always end at the same pressure level it began. For a male voice, you may count about 180 complete cycles in one second of audio, which would give the speaker's voice a pitch of 180Hz. A female singing voice may attain 600Hz. The key "A" below middle "C" on the piano is about 440Hz. Each time a frequency doubles, it is said to raise an octave. So, if 440Hz is "A" below middle "C", then 220Hz is "A" the next octave lower, and 110Hz is still "A", yet another octave lower. The high pitched ringing you may hear eminating from television sets is around 17,000 Hz. The human ear can perceive frequencies up to about 20,000Hz.

## Phase

As a wave cycles through at its particular frequency, it can be thought of as passing through all the degrees of a circle, from zero to 360 degrees. Each part of the cycle can be referred to by its phase, with zero degrees being the midline value as the pressure level is increasing, or as the wave is on its up-swing. Then 90 degrees would be the peak, and 270 degrees the valley. At 180 degrees, the wave is back at the midline, but this time on the down-swing. At any instant in time, a sine wave can have only one phase value. Phase is the one component that human ears can not discern very easily. For example, a bell rang at exactly 12:00pm, we would not be able to notice at exactly one second past 12:00pm whether the phase of the sound we are hearing is at 0 degrees, or 90 degress, or anywhere. If the bell rang at 3 milliseconds past 12:00pm, the phase at exactly one second past 12:00pm would certainly be different (depending on the pitch of the bell), but we still would not notice it was different.

The sounds we hear in everyday life and when we listen to music, are generally not just a pure sine waves as examplified above, but a collection of an infinite number of sine waves, each at its own varying amplitude, and its own varying phase. At any instant in time, what we truly hear is the sum of all of the frequencies present, each at their own amplitude, and each at a particular phase during their cycle. Summing all these will give exactly one value, or one pressure level that is present at the eardrum at any particular instant. Because of this, a wave can be graphed as the instantaneous amplitude (or pressure level) during an interval of time. Cool Edit's Spectral View will display all the frequencies present at any instant in time (almost), and their amplitudes at that instant. The louder a particular frequency is, the brighter the point will be. The higher up on the display the point is, the higher the frequency is that the point represents. Time is still represented as going from left to right. The reason I say the Spectral View *almost* displays all frequencies present at any instant in time is because it is impossible to determine all frequencies at any instant that went into producing the pressure level that was measured during recording, but it can be estimated from the audio before and after that point. This is why, as you zoom in more and more, the particular frequencies begin looking very smeared from left to right.

## Sampling



In the computer, it is impossible to work with an infinite amount of data, which is what would be required if a continuous wave were to be represented inside the machine, so at every possible instant in time we would have the value of the waveform at that instant. For this reason, it is necessarey to *sample* the data. Sampling consists of breaking up the waveform at constant intervals, and representing all values in that interval by a single value. By dividing the waveform up like this, one second of audio can now be represented by a finite number of values. The *sample rate* is the number of divisions taken per one second of audio. As you can see by the graph, the sampled waveform (horizontal dashes) contains much less information about the shape of the wave than the original continuous wave itself. The highest frequency that can be represented by this method is exactly one half the sample rate. So, if a sample rate of 22,050 slices per second was used, the highest frequency that can be represented would be a tone of 11,025 Hz. This frequency is known as the Nyquist frequency. The higher the sample rate, the higher the frequencies are that can be represented by sampling. The human ear can hear frequencies up to about 20,000 Hz (or 20 KHz), so to represent all sounds that humans can hear, a sample rate of at least 40,000 Hz must be used, which would yield 40,000 values for the computer for each second of audio. This is why CD players sample at 44,100 Hz, to be enough above twice the highest frequency anyone can hear.

So far we have broken *time* into discrete intervals. What about the actual values, or amplitude, at each of those intervals? The wave above can be thought of as going from values between 100 (at the top) to -100 (at the bottom). This would cause the short sequence of 11 samples to be given to the computer as roughly 93.7, 51.5, -22.1, -89.4, -97.6, -48.0, 25.7, 92.1, 93.9, 54.5, and -21.1. The same problem as with having continuous time exists with the values at each interval as well. There are an infinite number of possible values the wave can have during each of these intervals. For example, instead of the first value being 93.7, it could just as easily have been 93.716746352231... These values must also be broken up into intervals (which may or may not be evenly spaced) so that computers can deal with them. This breaking up of continuous values to a specific number of values is known as *quantizing*. An 8-bit sound card, for example, can have any one of 256 unique values, where the continuous range of real numbers has been quantized to 256 steps. The reason 8-bit sound has 256 possible values is because 2 to the 8th power (2 x 2 x 2... eight times) is 256. The 16-bit sound cards can have any one of 65,536 values for each interval. The quantization level, directly relates to the *dynamic range* (difference between the softest and loudest sound) that can be represented (if the range has been evenly divided into equal spaced steps). With only 256 levels, the sound quality is compareable to that of AM radio. With the 65,536 levels possible with 16-bit audio, the quality is compareable to that of compact discs, which can have much louder 'louds', and much softer 'softs'. So, where the sample rate (divisions in time) determines the highest frequency sound that can be represented, the quantization level (divisions in amplitude) determines the highest dynamic range that can be represented. If the divisions in amplitude are not evenly spaced, but there are more divisions with lower values, and fewer with higher values, softer sounds can be represented with fewer volume levels, at the expense of less precision at higher volume levels. The A-law and μ-law file formats do just this -- they save only 256 unique volume levels, but have more levels at the lower volumes. The result is the dynamic range equavilant to having 4096 volume levels (compareable to a high quality cassette), but with some distortion (like distortion heard on broadcast FM radio, but a little worse).

## Sound Reproduction

The hardware in your sound card has a special chip for converting the *analog* (continuous time) signal to *digital* (discrete or sampled) signal called an A/D converter. It also has the recripicol chip, which converts the digital signal from the computer to an analog signal that can be piped to a pair of headphones, or to your stereo system which is called a D/A converter. Even though the sampled signal has far fewer samples than a continuous time analog signal, the D/A convert can reproduce the original analog signal perfectly. This means that the only loss in going from the analog domain to digital, then back to analog again is the loss in higher frequencies (those above 1/2 the sampling rate), and loss in the dynamic range (depending on the quantization level used). Of course, with lower quality components, there will be loss in the A/D converter by the converter not perfectly converting the original analog signal to digital. Also, there can be loss in the pre-amplifier of the sound card after the signal has been converted from digital back to analog. What this means, though, is that it is perfectly safe to take any audio signal, convert it to digital, work with it in the digital domain (on the computer using Cool Edit), and convert it back to analog (play the sound). This means we are no longer restricted to the limitations of electronic components in filters, special effects boxes, etc.

## Quality Issues

As alluded to before, there are many sacrifices, gives, and takes that must be considered when going from the analog domain to the digital. With enough money, you can purchase a 'high quality' sound card that has excellent A/D and D/A converters, and high quality pre-amplifiers for getting the signals from the real world into the computer, and back out again. But no amount of money in the world will make a 16-bit sample have a higher dynamic range than 96 dB, or a wave sampled at 22,050Hz able to contain frequencies above 11,025Hz. The current sound cards, though, are about matched in quality between their analog components and digital specifications. That is, the components can handle frequencies up to 24 KHz, and have a signal to noise (S/N) ratios approaching 96dB, which matches the digital specifications of a 48 KHz sample rate, and 16-bit word size. Some high quality amplifiers on the market can amplify signals upwards of 30 KHz, and also have S/N ratios of around 110 dB, perhaps more. Even though the digital portion may be trivial (sampling at 60 KHz, and using a 20-bit word size), a sound card of this calibre may be larger than your PC! If the components in the sound card can not handle these higher specifications, then the extra space required by the PC to store larger samples (20 bits instead of 16 bits) and more samples per second (60,000 as opposed to 48,000) would just be wasted.

## Noise

In general, noise is the opposite of pure tones or sine waves. Instead of sound at exactly one frequency, noise consists of random collections of all frequencies. At any instant in time, any number of frequencies may be present, at any volume, at any phase. The effect is that of the static you hear when tuning *between* FM radio stations, or the sound of your TV when it is not tuned to any broadcast. See the Generate Noise function for examples of types of noise, and try generating some yourself. Notice that in the Spectral View, the entire screen is filled with colors -- with random splotches all over. Noise can be *colored* by filtering it, which reduces the occurance of some frequencies, or increases the occurance of others.

## Filtering

Filtering is one of the most popular uses for digital signal processors, or programs that modify sound data. Simply put, a filter just adjusts the volumes of specific frequencies, or adjusts the phases of specific frequencies. Many of the functions in Cool Edit are based on the Filter. The Filter function itself allows one to choose exactly which frequencies should be boosted, or cut. The Noise Reduction function dynamically cuts frequencies by differing amounts depending on how much of that particular frequency is present. It analyzes the frequencies present in the unwanted portion, and attempts to cut out these, and *only* these frequencies while leaving all the reast untouched. The Quick Filter function boosts or cuts a large range of frequencies, that range from generally low frequencies to generally high frequencies. Any of these filtering functions can be tested by generating white noise (which contains equal amounts of all frequencies), filtering it, and then viewing it in the Spectral View. Just filter half the waveform and compare the difference between the unfiltered data and filtered data to see exactly which frequencies have been boosted, or cut.

## The FFT

The Fast Fourier Transform is an algorithm that Cool Edit uses to perform its filtering functions, as well as its Spectral View and Frequency Analysis functions. This transform takes data in the time vs amplitude format (the Waveform View plot), and converts it to time vs frequency (the Spectral View plot). It can also convert back the other direction. For more detailed information on the FFT, see any books on Digital Signal Processing.

## Working with Files

Cool Edit can work with a variety of file formats. All file formats are converted to either 8 or 16 bit PCM audio while being edited, and written back in the original format when saved. The file being worked on is kept in a temporary file, so edits are not made to the original file until saved back. The following cover the items available from the **File** menu.

### CLOSE

Close will close the wave currently being edited, and show the initial startup information in the status boxes. When a waveform is closed, the associated temporary file is removed, thus freeing up the hard disk space that was previously being used.

### EXIT

Choosing Exit, or double-clicking Cool's close box will prompt to save the current file if it has been modified. Upon closing, the main temporary file is removed, and any audio playing or recording is stopped. When the last instance is closed, Cool ensures that all temporary files have been removed, including all undo buffers.

### NEW

When creating a new waveform, you must specify the waveform properties. Using higher sampling rates, stereo, or higher bit resolutions will result in higher quality sounds at the expense of requiring more memory.

**Sample Rate** This describes how many times per second to take a snapshot of the audio. The human ear can perceive sounds up to about 17,000 cycles per second, or 17 Khz. When choosing a sample rate, frequencies of up to 1/2 the sample rate can be produced effectively. So to reproduce frequencies up to 10Khz, a sample rate of at least 20Khz must be chosen. Choose **Custom** to enter a sample rate not listed.

| | |
|---|---|
| 11,025 Hz | Poor AM Radio Quality/Speech |
| 22,050 Hz | Near FM Radio Quality |
| 32,075 Hz | Better than FM Radio Quality |
| 44,100 Hz | CD Quality |
| 48,000 Hz | DAT Quality |

**Channels** Mono waveforms support one channel of audio information. Stereo files take twice the space because there have two channels of information represented, a left and a right channel.

**Resolution** This describes the number of bits to use for each sample on each channel.
**8-bit** resolution will provide 256 unique "volumes". The PC-Speaker, for example, provides only 4-bits of resolution because it can support 16 unique volume levels.

**16-bit** resolution will provide 65,536 unique "volumes", for roughly a 96 dB signal-to-noise ratio.

Much quieter sounds can be reproduced at 16-bit resolution than at 8-bit resolution, which only has about a 48 dB signal-to-noise ratio. The percieved signal-to-noise ratio, and actual ratio may both vary somewhat depending on the audio and recording method. Compact disk players have a 16-bit resolution.

Note: Certain combinations of sample rate, channels, and resolution may not be available on your system. To see the maximum capabilities of your system, look at the status window when starting the program. You can also see the capabilities of your sound board by viewing the **Select Waveform Device** dialog. You can still create, load and edit files that cannot be played on your system, you just may not be able to play them properly or at all.

## New Instance

Selecting New Instance will open another Cool Edit window as if you opened another instance from the Program Manager. Multiple instances is analagous to multiple waveform edit windows in other programs. Open more instances of Cool for any of the following reasons:

- Use other instances for temporary storage -- as a second or third clipboard.
- Mix several waveforms together by opening each source wave in its own window, then use **Paste Special** to mix into a final form.
- Test complex operations on an audio selection in a second window so as not to modify the original.
- And any more you come up with.

Many instances can be open at once using a minimum of extra memory per instance. The number of open instances of Cool Edit are only limited by your system memory and limitations of the Windows operating system being used.

## Open

Open a new file for editing. Various waveform data types are supported. Cool Edit will convert the waveform type to its own internal format for faster editing. The audio data size displayed by Cool may be larger that the original file if the file was in a compressed format. The wave can subsequently be saved in any of the supported formats you wish. By opening a waveform of one format and saving in another you can convert between any of the available formats. Mutiple files may be opened at once, but all will be converted to the format of the first file opened (alphabetically). If multiple files are chosen, the Cue List will contain range markers for each file.

Each wave type supported is embodied in a .FLT file filter file. I have provided the file filters for a number of popular formats. New formats can be added by following the File Filter Application Programming Interface (API). These .FLT files are just DLLs that have been given the .FLT extension. The DLL contains all the procedures necessary to read and write in the format they support.

| | |
|---|---|
| **Auto Play** | When Auto Play is checked, the wave will be played once it is clicked or typed in the File Name box. If the ACM driver (that is installed as a Windows driver and is not part of Cool Edit) does not support the wave format, then it will not play. |
| **Show Info** | When Show Info is checked, the waveform's information will appear at the bottom of the dialog. The top line displays the compression format and uncompressed file size. The bottom line displays the sample rate, bit size, number of channels and total playing time. |

Following are all the wave formats supported so far.

**<u>WAV - Windows PCM waveform</u>**

All .WAV files consist of a RIFF WAVE header, waveform data, and optionally more RIFF data (as can be entered in the waveform **Info** box under **Options**). The standard Windows PCM waveform data consists of Pulse Code Modulation waveform data, which is just the exact amplitude of each sample.

**<u>WAV - Microsoft ADPCM waveform</u>**

Microsoft ADPCM compressed waveform data consists of 4-bit per channel compressed data.. Each 4-bit sample is expanded to 16-bits (or cut down to 8-bits if needed) when loaded. This compression results in some signal distortion. Distortion is more intense in the higher frequencies (above 2KHz), but minimal in the lower frequencies.

**WAV - IMA ADPCM waveform**
This standard compresses 16-bit waves to 4-bit using a different (faster) method than Microsoft ADPCM, and has different distortion characteristics, which may be better, or worse, depending on the original sample being compressed. Distortion is less in the high end compared to Microsoft ADPCM, but is spread out evenly throughout the entire frequency range.

It is better to save 16-bit audio in this format than 8-bit. This format allows for 3-bit compression as well at a slightly lower quality. Very few sound drivers support the 3-bit ADPCM, and I have found none that actually work properly. In building this format, I followed the specification to the letter without making any assumptions. In the past, I have seen audio drivers that did not play DVI compressed data properly, but lately this has been changing as other manufacturers are providing DVI audio drivers that can read the files saved by Cool Edit in this format just fine. (Although I have yet to see a 3-bit DVI audio driver that plays stereo waves properly.) If you have other software that does not play files saved in this format properly, please contact the vendor and try to obtain the latest driver they have.

I have also implemented a 2-bit version of the compression by using the index table [-1,2,-1,2]. I could not find the specification for 2-bit compression, so it may not be compatible with other IMA/DVI 2-bit compressed files. I found that the preceeding index table provided the best quality.

**WAV - CCITT mu-Law and A-Law waveforms**
These formats compresses original 16-bit audio down to 8 bits. The quality is somewhere between 8-bit and 16-bit. Thus, a-law and mu-law encoded waveforms have a higher s/n ratio than 8-bit PCM, but at the price of a little more distortion that the original 16-bit audio. The quality is definately higher than you would get with any 4-bit ADPCM formats.

**VOC - Sound Blaster voice file format**
Sound Blaster and Sound Blaster Pro voice files will work with this format. Only 8-bit formats are supported. Maximum sample rate for mono is 44100 samples/second, and 22050 samples/second for stereo. Looped audio will be *unraveled* and silence space expanded when opening a file in this format. Cool Edit will only save straight audio data. In this format, no looping or silence encoding will be done. The only **Info** box information saved will be the Comments section.

**AIF - Apple AIFF format**
This is Apple's standard wave file format. Assuming you got the wave over to the PC, you can name the file with the .AIF extension, and load it using this file filter. I have tested it on several Apple sound files, and have been able to read them in OK. Macintosh computers should be able to read in AIF files saved in this format if the file type is changed to "AIFF" on the Macintosh.

**TXT - Text file format**
Data can be saved in a standard text file, with each sample value terminated by a line return, and stereo samples separated by a tab (as left_sample<tab>right_sample<cr><lf>). The sample value can be in either normalized (between -1.0 and 1.0) or original (-32767 to 32768) form. An optional header can be written which just gives the sample type of the original audio. If normalizing, the header is required. To see what the exact format is, simply save a short snippet of audio as Text and view it with Notepad. This format is handy for importing and exporting data between Cool Edit and numerical analysis programs, spreadsheets, etc.

The header is formatted as KEYWORD:value with the keywords being: SAMPLES, BITSPERSAMPLE, CHANNELS, SAMPLERATE, and NORMALIZED. The values for NORMALIZED are either TRUE or FALSE.

For example,

```
SAMPLES: 1582
BITSPERSAMPLE: 16
CHANNELS: 2
SAMPLERATE: 22050
NORMALIZED: FALSE
164 <tab> -1372
492 <tab> -876
...
```

**AU - Next/Sun CCITT mu-Law, A-Law and PCM format**
The Next/Sun format supports many data formats, but only mu-law, A-law and linear PCM data are supported. The most common use for the AU file format is for compressing 16-bit data to 8-bit mu-law data. See the the WAV CCITT format for more information on mu-law and A-law.

**SMP - SampleVision format**
The SampleVision format only supports mono 16-bit audio. If your data is in a different format, you will be asked if you want to convert it before saving. This format supports loop points, which can be edited using the Cue List. The Label of the cue must be in the format Loop n,m where n is the loop number from 1 to 8, and m is the mode being 0=no looping, 1=forward loop, 2=forward/back loop. The Play List is used to enter the number of times to loop the cue range. Add the cue range to the Play List, then enter the number of times to loop.

**VOX - Dialogic ADPCM**
This is yet another 4-bit ADPCM file format. It has been optimized for low sample rate voice, and will only save 8000Hz Mono 16-bit audio. This format has no header, thus any file format with the extension VOX will be assumed to be in this format.

**SAM - 8-bit Raw signed format**
This format is similar to 8-bit AIFF, except that there is no header written. When saving 16-bit files in this format, they are automatically converted to 8-bit before being written out. In order to read 8-bit signed raw data, the filename must have the extension ".SAM". This format is handy for reading and writing MOD compatible waveforms.

**PCM - Raw PCM Data**
This format is simply the dump of the PCM waveform data without any header information. None of the waveform **Info** box information will be saved. Since there is no header, you will be asked to select the waveform sample rate, resolution, and number of channels when loading. By reading audio data in as PCM, practically any audio file format may be read in! Once read in, you can use the **Adjust Sample Rate** and **Byte Swap** functions to modify how the file is interpreted until you have something that sounds like real data. The waveform will sound bad in different ways depending on which parameters are incorrect. Once the wave is loaded, and sounds almost all the way fine, you may hear clicks at the start, end, or sometimes throughout. The clicks are various header data being interpreted as a wave. Just cut these out, and *Voila*! You have read in a wave in an unknown format!

**MPG - MPEG Layer 2**
The MPEG (Motion Picture Experts Group) format, layer 2, is available upon request. This format will allow the saving of CD quality (16-bit 44.1khz stereo) audio in 1/4 the space with no noticeable loss in quality. Higher ratios, like 10:1 are attainable with very little loss in fidelity.

## OPEN AS

This works just like Open except that the sample format can be specified first. If the sample format selected is different than the native format of the wave being opened, then it will be converted, using the "quick-and-dirty" conversion method. This means that if the target sample rate is different, the conversion will not try to pre-filter or post-filter the samples, which will cause some non-linear distortion. If the wave is just converted between number of channels or sample size though, it is the same as using the Convert Sample Type with default parameters.

After choosing the filename, choose the new sample type. The waveform will be converted as it is being opened.

## OPEN APPEND

This works just like Open except that the file is appended to the end of the file currently being worked on. If thie file being opened is in a differing file format, the "quick-and-dirty" method of file conversion is used to conver the sample to the same format.

## RE-OPEN

This will reload the previously loaded waveform, replacing any changes that were made to the waveform being edited. Be aware you will lose any changes you have made since the last time the wave was saved. There is no prompting to save modifications since it is assumed you wanted to re-open the waveform and disreguard any of your changes.

## SAVE

Choosing Save from the File menu will save the current waveform back to disk, overwriting the original without confirmation. The file format will be the same as the original format of the file. For example, if an ADPCM compressed file is loaded and modified, then Save will save it back in ADPCM compressed format.

The disk icon in the toolbar can be set to save immediately, or to bring up the **Save As** box depending on the setting in the **Settings** dialog.

## SAVE AS

Save As brings up the Save File dialog and allows you to choose the file format you wish to save your waveform as. The waveform is saved in the current sample format (sample rate, bit size, and number of channels). See **Open** for a list of file types. The name of the current waveform assumes the name of the Saved As file.

## SAVE SELECTION

Saving a wave selection is identical to Save except that only the highlighted selection is saved, and not the entire waveform. Also, the name of the current waveform is not changed to the name of the file being saved.

## General Purpose Editing

A host of editing commands are available. Some have further options giving you complete control over general purpose editing. Audio can be copied from and pasted to any number of currently running instances of Cool. If the various instances have audio of differing sample rates, bit rates, or channels then the audio will be quickly converted (with low quality sample rate conversion) automatically. The following cover the items available from the **Edit** menu.

### ADJUST SAMPLE RATE

You can change how Cool Edit interprets the actual waveform data by adjusting the sample rate, bit rate, and number of channels. The actual data is left untouched; only the way the data is interpreted changes. Be aware that listening to Mono data in Stereo format, or listening to 8-bit data in 16-bit format will sound really really weird! If you want to convert the current waveform to a new sample type without affecting how it currently sounds, use **Convert Sample Type** function instead, since this operation only modifies how Cool Edit intereprets the data.

Adjusting the various parameters comes in very handy when loading in waveforms of unknown type (RAW). You can play with the various settings until the wave sounds "right".

### BYTE SWAP

You can change how Cool Edit interprets the actual waveform data by swapping high and low order bytes in 16-bit mode, or by adjusting the signed/unsigned interpretation of bytes in 8-bit mode. Swapping the byte ordering comes in very handy when loading in waveforms of unknown type (raw PCM). You can try swapping the bytes if the waveform does not sound correct after loading an unknown format.

### CONVERT SAMPLE TYPE

Convert the currently loaded waveform to a new sample type. Unlike **Adjust Sample Rate**, this will do all the necessary conversions so the sample sounds the same at the new sample type. When converting to a new sample type, various other options may be avialable so that you may customize the conversion process.

First choose the new Sample Rate, number of Channels, and Resolution you would like to convert the loaded waveform to. Then you may choose some of the following options:

**Low Quality** When converting to a new sample rate, choosing Low Quality will do the quickest conversion, with no pre- or post filtering. This quick conversion may cause side effects such as “ring” or aliasing in the higher frequencies.

**High Quality** Choose this when you want to cleanly convert to a new sample rate. Proper filtering is done so as not to introduce any aliasing or ringing. Use this option if you are willing to wait a bit longer for a *quality* sample.

**Emphasis** When converting to a lower sample rate, there is a noticeable loss in high frequencies. To compensate for this, you can choose a high end emphasis of 2dB or so. Frequencies from 1/4 up to 1/2 the converted sample rate will be boosted. Choosing a Low Quality conversion will sound as if it is keeping some of the higher frequencies, but it also induces extra artifacts which vary depending on the ratio of original and converted sample rates and the type of audio being converted. Choosing a 0dB emphasis will do no extra emphasis on the high end.

| | |
|---|---|
| **Left Mix** | If converting to Stereo, you may choose the amplitude at which the original signal will appear |
| **Right Mix** | in the new stereo sample. This way, you can place the Mono signal on the left channel only, the right channel only, or any volume level on each channel you wish. If converting to Mono, these percentages control the amount of signal from the Left and Right channels respectively to mix into the final mono signal. The most common mixing method is to use 50% of both left and right channels when converting to Mono, and 100% for both values when converting to Stereo. To do a *vocal cut*, you could convert a stereo waveform to mono with a Left Mix of 100% and a Right Mix of -100%. For an interesting "inside your head" effect, you could convert a mono waveform to stereo with the same Left Mix of 100% and Right Mix of -100%. |
| **Dither** | If converting 16-bit audio to 8-bit, you have the option to dither the audio, which allows you to retain some quieter audio at the expense of some low level noise. Without dithering, quiet audio will just crackle and pop. If dithered, the same quiet audio can be heard distinctly behind some hiss. |

## COPY

Waves can be copied to the clipboard to be pasted later into Cool Edit or any other application that supports the CF_WAVE clipboard wave format. Before copying, the portion that is to be copied must first be selected.

Extra large waves may not have enough free memory to be copied. If this happens, try breaking up the selected range into smaller portions and copy/paste them in multiple steps, or save the selection out using **Save Selection**, then use **Paste Special** and choose From File to paste the audio that was just saved back in.

Use CTRL+C or CTRL+Insert to quickly copy the wave.

## CUT

Cut will copy the selected range to the clipboard, and remove it from the waveform being edited. Once in the clipboard, it can be pasted or special pasted to waveforms, or pasted to other applications that support the CF_WAVE clilpboard format. The same limitations apply to Cut as do to Copy.

Use CTRL+X or SHIFT+Delete to quickly cut the selected wave.

## DELETE SELECTION

Once a range is selected, it can be removed by choosing this option. The deleted portion is not copied to the clipboard. It is gone forever (unless Undo is active).

Use the DEL key to quickly delete a selection.

## EDIT LEFT/RIGHT

For stereo waveforms, it is sometimes handy to be able to work with a single channel. Normally, both Edit Left, and Edit Right are checked, meaning both channels are being edited simultaneously. To edit only one channel, un-check the channel you want to preserve, so only the other channel is being edited. When pasting, the audio data is overlapped with what is already there, since inserting only on one channel will place the two channels completely out of phase (if this is your desired effect, use the **Delay** function). Many of the Transform functions will operate on a single channel of a stereo waveform.

## Paste

Paste will insert the wave from the clipboard at the current insertion point, replacing any waveform data in the selected range. If the format of the waveform data in the clipboard differs from the format it is being pasted into, it will be converted accordingly before pasting occurs. If no selection is highlighted when Paste is issued, the clipboard data is inserted at the insertion point.

Use CTRL+V or SHIFT+Insert to quickly paste the waveform that is on the clipboard.

## Paste Special

Waves from the clipboard or from a wave file can be looped, or mixed with the current waveform. They are inserted, overlapped, or modulated starting at the current insertion point. Paste Special is a very powerful paste function and can be used in a variety of situations.

| | |
|---|---|
| **Volume** | Use the volume slides to paste an amplified version of the clipboard wave into the current waveform. By adjusting the volume slides, single channels may be pasted. |
| **Lock left/Right** | When checked, the volume slide bars are locked, so both left and right volumes can be adjusted at the same time. |
| **Overlap** | When overlap is checked, the clipboard wave does not replace the currently highlighted selection, but is mixed at the specified volume with the current wave. If the clipboard wave is longer than the amount selected, the wave continues being pasted beyond the selection. |
| **Modulate** | When modulate is checked, the clipboard wave is modulated with the current wave. This is like overlapping, except that the values of the source wave and clipboard wave are multiplied by each other sample by sample (instead of added). To quickly modulate by a sine wave, use the Generate Tones function which has a “Modulate by Source” option. |
| **Loop Paste** | When checked, the clipboard wave is pasted the number of times entered. For effects, this gives a great “Max Headroom” stuttering effect. |
| **From Clipboard** | When chosen, the audio data to be pasted is the data currently on the clipboard. If the format of the waveform data in the clipboard differs from the format it is being pasted into, it will be converted accordingly before pasting occurs. |
| **From File** | When chosen, a file may be chosen with choose file to be pasted. This is especially useful when the amount of data you wish to paste is too large for the clipboard. Simply save the data you want to mix to a file, and paste in that file. |

## Repeat Last Command

Pressing F2 or choosing Repeat Last Command will bring up the settings dialog for the last command issued if a settings dialog exists, otherwise the command is carried out instantly. To carray the last command out instantly at any time, press the F3 key.

## Select Entire Wave

This will select the entire waveform from sample zero to the end of the waveform. It makes no difference if the view is zoomed in or not. Double-clicking on the waveform selects only the visible portion of the wave, while choosing Select Entire Wave (or pressing CTRL+A) will select the entire waveform independent of the portion being viewed.

## SPECTRAL/WAVEFORM VIEW

Choosing **Spectral View** will display waveforms by the by their frequency components. This is a handy function for analyzing your audio data, to see which frequencies are most prevalent throughout your data.

The more abundant a frequency is, the brighter the color displayed will be. Colors range from dark blue (next to no frequencies in this range exist) to bright yellow (frequencies in this range are very strong). I personally prefer this method to other "3D" frequency plots because so much more information can be displayed at once. At a resolution of 10 bits, spectral lines are clearly visible, and individual notes distinguishable.

Lower frequencies are displayed near the bottom of the display, while higher frequencies are displayed near the middle on to the top. The vertical display is linear. White lines on the left and right divide the display into 1/2, 1/4, 1/8, and so on. To top of the display represents frequencies at just below the Nyquist frequency, or 1/2 the sample rate. So if a bright spot appears near the top of the display for a signal sampled at 22K, the frequency being represented is near 11K. A bright spot at 1/4th of the way up would be 1/4th of 11K, or about 2.75K.

The vertical resolution of the spectral plot can be adjusted in the **Settings** section by adjusting the **Spectrum Analyzer Resolution** values from 5 (very course) to 12 (very fine). The finer the resolution, the longer it will take to display a frequency plot. All editing can be done while viewing in spectral mode insted of waveform (amplitude plot) mode, but on most machines drawing could take a while because of the enormous amount of calculations needed to compute the graph. It is best used on slower machines as an analysis tool only.

Choose **Waveform View** to return to original waveform view mode.

## TRIM

Trim is the exact inverse of delete, which means everything is deleted except the portion that is selected. Only the selected portion is kept. This is handy to quickly pick out the part of a recording you want to keep. If the Undo feature is enabled, and you are trimming out a very small portion of a very large waveform, there may be a considerable delay while the entire waveform is copied to the undo buffer. In cases like these, where you know you will just be trimming waveforms, turn off the **Undo** feature in the **Settings** dialog.

Use CTRL+T to quickly trim the selected portion of the wave.

## UNDO

If the Undo function is enabled, (see the **Settings** dialog), you will be able to back up a maximum of three steps if you make a mistake. Undo information is automatically saved in a temporary file called ~NDOnnnn.TMP in the undo temporary directory before anything happens that will change the waveform being edited. This could slow down wave operations on very large waves. The saving of undo information happens just before the progress meter begins displaying the progress of your action. If this is too long of a wait, and you have no need for undo, then disable this feature during those cases.

You will be warned if there is not enough disk space to save the Undo information before continuing any wave operation if Undo is enabled.

## VIEWING RANGE

Selecting **Viewing Range** or double-clicking on the green/black samples portion bar will bring up the viewing range window.

Enter the leftmost and rightmost samples that you wish to have displayed.  You can use this function to highlight a specific number of samples by double-clicking in the wave editing area after selecting the viewing range.

## Waveform Transformation Functions

Cool Edit has a great number of functions, and many functions serve several purposes. Whatever work you intend to do with a waveform, you should be able to find a function here to do the job, or enough tools to create the desired effect. The following cover the items available from the **Transform** menu.

### AMPLIFY

Amplify will increase or decrease the volume of the selected sample. The increase or decrease can change linearly, or exponentially over the length of the sample. The values of the starting volume and ending volume can be entered independantly for each channel, and vary over time in an steady incremental fashion in either the decibel or percentage scales.

| | |
|---|---|
| **Initial Amplification** | This is the amplification that will affect the beginning of the selection. Choose a separate final amplification for fading up/down effects. An amplification value of 100 will keep the signal unchanged. |
| **Final Amplification** | This is the amplification that will affect the ending of the selection. Setting both the initial and final amplifications to the same value will amplify the entire selection the same amount. |
| **Lock Left/Right** | Left and Right channels may be amplified at separate values. If the Lock is checked, then the scroll bars for the left and right channels are locked to the same value. Effects such as panning from left to right can be achieved by choosing separate values for the left and right channels. |
| **DC Bias Adjust** | Adjust the waveform so it is centered on the center line (0 %). If samples are recorded with a DC Bias, they will appear to be above or below the center line. They must be centered before doing other waveform transformations, and choosing this will center the wave properly. To skew the entire selected waveform above the center line, enter the percentage to move the waveform up in the adjustment box. For example, 50% will move the entire waveform up half way, and a - 50% will move it down half way. |
| **Logarithmic Fades** | If checked, fading is accomplished by exponentially adjusting the volume. Otherwise fading is done in a linear fashion. Logarithmic fading adjusts the volume of the sample from the initial amplification value at the beginning of the sample to the final amplification value at the end in constant increments (or decrements) in decibel volume. Linear fading does the same, except the increments are constant in percent volume. |
| **dB Scale** | When checked, initial and final amplification values are viewed as decibel increases or decreases from current volume. If not checked, amplification values are viewed as the percent of original amplification. Thus, a value of 200% is equivalent to about 6 decibels, or twice as loud. A 20 decibel increase is the same as a 10 times increase in volume. |

| | |
|---|---|
| **Normalize** | Pressing the Normalize button will calculate the greatest amplification for the sample that will *not* result in clipping when set to 100%. If the left and right scroll bars are not locked, separate left and right values will be computed, potentially amplifying one channel more than the other. To normalize to less than the maximum range, enter the percentage of maximum to normalize to. For example, choosing 50% will compute values needed to amplify the file no more than 50% of maximum, resulting in a 3dB diminuation from maximum output. If two sounds normalized to 50% are overlapped, the resultant wave is guaranteed not to exceed the boundaries, and will not clip. |
| **Presets** | **Center** - Uses the DC Bias adjustment to center waveform vertically<br>**Fade In** - Linearly fades out wave from full volume to silence<br>**Fade Out** - Linearly fades in wave from silence to full volume<br>**Pan L->R** - Fade out left channel while fading in right<br>**Pan R->L** - Fade out right channel while fading in left |

## CHANNEL MIXER

With stereo waveforms, the channel mixer allows one to have total control over the mixing between the left and right channels. The default values will leave the wave unchanged.

**New Left Channel** The slide bars give the percentage of each channel, left and right, that will go into the final wave after mixing. Choosing an L of 0, and an R of 100 will make the left channel equal to the right channel.

**New Right Channel** These two slide bars do the same as above, except they work on the right channel.

**Invert** Choosing invert for either channel will invert the channel. Peaks become valleys, and valleys become peaks. By inverting both channels, there will be no perceived difference in sound when listened to. But, inverting only one channel will greatly change the sound when listened to.

**Presets**

| | |
|---|---|
| **Vocal Cut** | This will sum the left channel with the inverse of the right, and place the result into both channels. On music where the vocals are heard equally loud on both channels, the vocals will disappear, or come close to disappearing. |
| **Swap Channels** | The left channel is placed on the right and vica-versa. |
| **Inverted Average** | The left channel is the average *(L+R)/2* of the two channels, and the right channel is the inverse of the average of the two channels -*(L+R)/2*. |
| **Average** | Both channels are set to the average of the two channels. |
| **Both = Left** | Both channels are set to the left channel (the left channel is copied to the right). |
| **Both = Right** | Both channels are set tot he right channel (the right channel is copied to the left). |

By playing with the combinations, effects of swapping channels, creating a mono sounding wave that is equal to the left, right, or a mixture of both channels, and creating waves whose left channel is the inverse of the right can be done.

## COMPRESSOR

The compressor function varies the output level based on the input level. This allows one to expand or compress the dynamic range of a sample, limit the dynamic range so all audio is at roughly the same level, or create a noise gate where all audio below a certain level is clipped to zero. This is all accomplished by use of a transfer function that is drawn using the graph. The graph depicts input level along the x-axis (left and right) and the new output level along the y-axis (up and down). A line from lower-left to upper-right (default) leaves the signal unchanged, since every input value goes to the exact matching output value. Other weird transfer functions can be drawn as well, for example, boosting all input that has a level of around -20dB, and leaving everything else unchanged. Or, drawing an inverse line (a line from upper-left to lower-right) will dramatically boost low amplitudes while dramatically supressing high amplitudes, that is, all quiet sounds will become loud, and all loud sounds will become quiet.

| | |
|---|---|
| **Attack** | The attack time determines the time it takes for the new output signal to reach the proper output volume. For example, if there is suddenly a quiet portion that drops 30dB, it will take this much time before the output actually drops to its corresponding volume level. If the sum of Attack and Release times is too short (less than about 20 ms total), audable effects can be heard, such as a "vibrating" sound at a frequency of 1000/<time>. So if the Attack and Release times are each set to 5 ms (making 10 ms total), then a vibrating sound at 100Hz can be heard. Thus, a total value of about 30 ms is about the lowest you can go without getting these effects. |
| **Release** | This is the time it takes the end of a previous output level to reach the proper output volume. For example, where the Attack is the time it takes the start of a pulse to reach the desired output volume, the Release is the time it takes for the end of the puls to reach the desired level. |
| **Samples/Group** | This is the number of audio samples to group together into one volume level change. A value of 1 is the best, so each sample gets it's own volume change. Larger values will change that many samples together at a time. You can use larger values without noticeable changes in quality. The only reason for using larger values would be for speed though, as larger values calculate much faster. You can use larger values for previewing how a compressor is going to sound, then Undo, and use a value of 1 when the compressor is set just the way you want it. |
| **Joint Channels** | In Stereo, each channel can compress independantly, sometimes causing the surrounding background noise to get louder on one channel at a time, which may sound strange. For example, a loud drum beat in the left channel will make the background noise sound louder in the right than in the left. If Joint Channels is checked, both channels are used to find a single input dB value, and both channels are amplified the same amount, together. For example, a loud drum beat on the left channel will cause the right channel to go quieter as well if compressing. |
| **Invert** | The invert button will change the graph to one that will function as the exact opposite. For example, if a transfer function with a compressor characteristic is being displayed, pressing Invert will change the graph to one with the corresponding expander characteristic. For a graph to be invertable, it must have pointes in the two corners (-100,-100 and 0,0) and it must be always |

increasing in output (ie,  you cannot go down in output volume as you go from left to right).   All segments must be sloping upwards from left to right.

**Presets**

**2:1 Compressor < 20dB** - Leaves signals above -20dB untouched, and compresses those below 20dB at a ratio of 2:1. Use this to make quiet passages more audable.
**3:1 Compressor < 10dB** - Leaves signals above -10dB untouched, and compresses those below 10dB at a ratio of 3:1. Use this to more drastically make quiet passages louder.
**3:1 Compressor > 30dB** - Leaves signals below -30dB untouched, and compresses those above -30dB. Use this to quiet loud peaks in a signal.
**3:1 Expander < 10dB** - Leaves signals above -10dB untouched, and expands those below -10dB at a ratio of 3:1. Use this to make quiet sounds much quieter. Expanders work well as hiss reducers in speach passages.
**4:1 Compressor > 20dB** - Leaves signals below -20dB untouched, and compresses those above -20dB. Use this also to quiet loud peaks in a signal, but leave more of the quieter signals untouched.
**Limiter 20:1 @ 0db - 15dB** -All audio is limited to the range -15dB to -20dB. Use this to make all audio about the same volume, regardless of original input volume.
**Noise Gate @ 20dB** - All audio quieter than -20dB begins to get clipped to zero volume. Before silencing, a 10dB buffer range is used to ease the transitions into and out of silence. So, all audio quieter than -30dB is clipped to absolute silence. Use these noise gates to replace noise between spoken words with absolute silence.
**Noise Gate @ 30dB** - All audio quieter than -30dB begins to get clipped to silence, and all audio quieter than -40dB is clipped to absolute silence.

**Compressors** are used for the compression of the dynamic range of an audio signal. It is generally an amplifier with two gain levels: the gain is unity for input signal levels below a certain threshold, and less than unity for signals with levels above the threshold. Compressors can be used to eliminate the variations in the peaks of an electric bass output signal by clamping them to a constant level, thus providing an even solid bass line. To maintain the original character of the instrument it is necessary to use a compressor with a long A/R time compared to the natural decay rate of the electric bass. Compressors can also be useful to compensate for the wide variations in the signal level produced by a singer who moves frequently, changing the distance from the microphone.

**Limiters** are compressors with a compression ratio of 10:1 or greater because their output levels are essentially clamped to the threshold level. A limiter can be used to clamp all audio to a prescribed output level, or just all audio above a certain threshold.

**Expanders** are used to expand the dynamic range of an audio signal, opposite of the compressor. It can also be considered an amplifier with two gain levels: the gain is unity for input signal levels above a certain threshold, and less than unity for signals with levels below the threshold. The expander is used to expand the dynamic range of an audio signal by boosting the high-level signals and attenuating the low-level signals.

**Noise Gates** are a special type of expander that can be used to reduce noise below a threshold level. It attenuates heavily signals with levels below the threshold. It is used to totally cut off the signal level during a musical pause so as not to pass the background noise present. It can also be used to silence the pauses in speach.

## DELAY

Delay either channel up to 100 milliseconds or more with the option to mix in the original signal with the delayed signal. Great for effects such as spatially locating a previously mono wavesource to the left or to the right, so that the sound will appear to emanate from that direction when listened to with stereo headphones. Delays of longer than 50 ms may be entered for creating a single echo.

**Delay** The actual amount of time to delay the channel in question. This value is in milliseconds. A delay of 1ms is about the time it takes audio to reach one ear before the other if coming from the side.

**Mixing** You may choose to have the resultant wave the delayed signal, keep it the original signal, or mix the two. A value of 50 will mix the two evenly.

**Invert** The delayed signal may be an inverse of the original if this is checked. More special effects!

**Presets** **Tunnel -** This preset can be used with mono as well as stereo waveforms. The settings provided give a nice tunnel/tubular effect.
**Spatial Echo -** With stereo waveforms, this gives a nice fake "big room" effect.
**Spatial Left -** If a mono wavesource was converted to stereo (so that the left and right channels are the same), then choosing this will make the sound appear as if it is coming from the left, since the right channel is delayed just enough so your brain interprets the sound as coming from the left. You must use headphones to hear the effect.
**Spatial Right -** The same as Spatial Left, but it locates the sound to appear as if it is coming from the right.

## DISTORTION

The Distortion function will map any sample value to a new sample value based on the transfer function graphed. The sample value is described in terms of decibels, the loudest sample value (highest absolute value of the sample) equates to 0dB, while the quietest sample value (values closest to zero) are seen as -48dB for 8-bit audio, or -96dB for 16-bit audio.

An upward sloping diagonal line from the lower-left corner to the upper-right corner in the graph represents unity, where each sample gets mapped (or is replaced with) itself, thus not changing the audio at all. Any deviation from this will distort the samples by replacing them with a different value. Try adding a point in the middle of the graph, and drag it upwards or downwards to change the graph from a pure diagonal line to induce some distortion.

Some distortion effects can give the impression of the audio being played on a car stereo too loud, on an old fashioned amplifier too loud, etc. Other effects possible are grunge or metal guitar sounds from a regular acoustic guitar, or just totally distorted audio that sounds so bad, that... well... it just sounds really really bad.

## ECHO

This function creates a continuous echo, with each echo equidistant apart. Each successive echo decays in amplitude by the falloff ratio. To create the effect of a single echo, use the Delay function instead. For great special effects, each echo can change in tonal quality by slightly equalizing each successive echo. Other interesting "feedback" effects can be created by choosing extremely small Delays (1 ms or so) and large Ratio values (like 99%).

| | |
|---|---|
| **Ratio** | Each successive echo's volume will be a certain percentage less than the previous one's. Choosing a falloff ratio of zero would result in no echo at all, while choosing a ratio of 100 would produce an echo which never gets quieter. For that Reverse Echo effect, simply reverse the waveform first, then echo, then reverse again. |
| **Delay** | This is the number of milliseconds to place between each echo. A delay of 100 milliseconds is equivalent to a 1/10th of a second pause between echoes. Choosing very small values of delay produces quite interesting effects, while extremely large values (above 1000 ms) produce Grand Canyon style effects. |
| **Initial echo volume** | This is the volume at which the echoes will be mixed with the original sample. Choosing smaller percentages (30% or so) is nice if the effects of the echoing at 100% make the sound incomprehensible. If Echo Bounce is chosen, try a lower initial echo volume for a single channel (for example, left of 10, right of 100) to increase the bouncing back-n-forth effect. |
| **Continue Echo** | Choosing Continue Echo will echo the highlighted selection over the rest of the unhighlighted area, stopping at the right-hand edge of the wave that can be seen in the window. If the window is zoomed in, the echoing will stop before the end of the file, since it will stop at the right hand side of the portion on screen. By using this option a single word, for example, can be highlighted and echoed over other audio, without echoing the other audio as well. |
| **Echo Bounce** | Selecting this option will make the echoes travel back and forth between the left and right channels. |
| **Equalizer Bars** | The echo "quick filter" lets you choose approximately which frequencies get removed from the echo first. A setting of zero will leave the frequency band unchanged. You can choose the frequencies that are "absorbed" as the echo progresses. The echoed sample is re-filtered through the quick filter on each successive echo. Setting all values to zero turns off the equalization, since no frequencies are to be absorbed. A single frequency slider actually controls the labeled frequency, and all others below it (it's not a true equalizer, but is great for creating effects). |
| **Presets** | All the echo presets are pretty much self descriptive. Try them out, and then create some of your own! |

If you wish to echo the right channel only, select an initial echo volume of 100% for the right, and 0% for the left.

## 3D Echo Chamber

This function will calculate the actual echoes as if the source audio (highlighted selection) and microphones (destination channels for echoed wave) were in a room of any given size and with walls of any given dampening factors. The number of echoes to calculate is adjustable, up to about 5,000 echoes. The more echoes there are to calculate, the longer it will take the function to complete. Practically any "ambiance" setting can be created using this function, from a small intimate room to a giant hall, to anything inbetween like a parking garage, underground pipe, or even a cathedral. The more echoes you choose to generate, the longer it will take to calculate them all, and perform the echoing. Since the echo function works in two stages (calculating echos, and performing echoes), don't be alarmed if the progress meter doesn't show any progress for the first few seconds for high numbers of echoes.

One great use for this function is to convert Mono audio to Stereo with all the right ambiance. Choosing a "left" microphone that is one to two feet away from the "right" microphone will simulate the ears of a listener, and will give the effect of "being there" when listened to with stereo headphones. Be sure to copy the mono audio into a stereo format before performing the echo so you can choose two separate microphone locations. A spatial stereo expansion effect can be created by placing the two microphone locations far apart, further apart in the settings than you will be playing them through speakers in real life. For example, if your stereo speakers are 6 feet apart, try placing the left and right microphones 20 or 30 feet apart in the settings.

To give more control over the environment, dampening factors can be applied to any of the 4 walls, floor, and ceiling. If a wall has a dampening factor of 1.0, it is totally reflective (like cement). If a wall has a very low dampening factor, like 0.05, it will absorb most of the sound (like carpeting or sound proofing panels). You can also lower the dampening factor of some of the walls to simulate the fact that other objects in the room are absorbing some of the audio.

Always place the microphone(s) sufficiently far from the source. If the microphone and source are too close together, you will just hear the source and no echoes since it is analgous to placing your ear right next to the sound source where you hear the sound only (which is very loud) and nothing else.

| | |
|---|---|
| **Room Size** | The length, width, and height of the room can be entered in units of feet (sorry, no metrics this time... There are approximately 0.3 meters per foot for those who need to convert). When entering source and microphone locations, they must lie between zero and the room's width for the "Distance from Left" parameter, and zero and length for the "Distance from Back" parameter. Room sizes can be as large as memory will allow. |
| **Intensity** | The volume of the echoes is determined by the volume of the first (direct) audio. The direct sound that reaches the microphone from the source will be at the same amplitude as the original audio being echoed. Thus, in a room of any size, if all 6 dampening factors are set to zero, there will be no change when echoing. Every echo adds to the amplitude of the finished audio, so the intensity should be set to less than 100%. In fact, the more echoes there are, the lower this value should be set to prevent clipping. In general, about 1% less for every echo, down to a reasonable limit of about 5% for 100 echoes or more (since after a while, the echoes are so quiet, they don't add to the final amplitude of the audio). |
| **Echoes** | This is the number of actual echoes to produce. To get a nice reverb and ambiance effect, at least 30 echoes should be generated. The more echoes that are generated, the truer the result will sound. You must sacrifice the quality you desire with the time you are willing to wait for the final product. Generate about 15 echoes or so to test the chamber size and general room sound, then increas that dramatically for the final production. Up to 4,000 echoes can be generated for the *very* very patient. |
| **Damping Factors** | Use the damping factors to set the type of room in which the audio is being played. The factors can simulate wall coverings, floor coverings, and other objects in the room that absorb sound. Granted, in real life, various objects absorb different frequencies of audio. In this simulation, all frequencies are reflected equally. The effects of speaker placement enhancing or cancelling certain frequencies, though, is still accurate. The fact that cement reflects high frequencies better than low ones is not accounted for, but great effects can still be achieved, and these effects are much more realistic than the basic **Echo** function. A damping factor of 1.0 is the greatest, simulating total reflectivity. A factor of 0.0 is the lowest, for total sound absorbtion by the reflecting surface. If enough people want the walls to absorb various frequencies, and not just *all* frequencies, I can add the parameters if enough people want it. |
| **Source Signal Placement** | The source (highlighted audio before running this function) can be placed anywhere in the room. The audio is simulated as a point source of audio, not directional. This means the audio will radiate outwards in all directions from the source, and not more in one direction than another. Directional audio would require a handful of new parameters and if this is desired, I can add it in if enough people say they want it. The distance the source is placed to any of the walls will affect the frequencies that are enhanced. In otherwords, source signal placement is crucial to the ambiance effect that is gained with this function. With stereo source, each channel can be placed independent of each other. |
| **Microphone Placement** | There can be up to two virtual microphones. Each microphone represents a destination audio track. The audio placed back into the waveform (the result of the echoing) is exactly what the microphone would hear if it were in the room at the location specified. Stereo signals have two pick up microphones while mono signals have only one, since there is only one channel in which to place the result. Placing the microphones in a |

stereo setting one foot apart will simulate the ears, and when listened to with stereo headphones, will sound as if you were actually in the room (if enough echoes are generated).  The brain will be able to pick out the directions of each echo, as well as the fact that the delays of the echoes will give the brain cues as to the size of the room. Placing the microphones very far apart and listening with headphones will give a very large "aural" or "Spacy" feeling to the audio, like it is all around you and inside you. Don't place the microphones too close to the source, otherwise the relative volume of the echoes will be so low that they will not be able to be heard.

| | |
|---|---|
| **Mix Left and Right Into Single Source** | When working with stereo audio, there are actually two source signals, one for each channel that can be placed independently. This takes twice as many calculations as a single audio source, so this option allows you to mix the left and right into a single point source for faster calculations. |

## ENVELOPE

By using an envelope, you have greater control over which parts of your wave are amplified, and by how much. A point at the top of the graph represents the maximum amplification, while a point at the bottom represents zero amplification (or silence). This function is handy when modifying Tones generated with this program to create more realistic sounding instruments and effects. For just decreasing the volume of extra loud peaks or increasing the volume of quiet portions in a waveform, the **Compress** function will do a better job than "eyeballing" the peaks and running an amplification envelope over them.

| | |
|---|---|
| **Amplification** | Adjust this value to change the maximum amplification value. This value changes the values represented by the graph. A value of 100% is the default. |

The graph is a Time versus Amplitude plot. (Time runs left to right and Amplitude up and down). Click anywhere on the graph to create a new point. Points on the graph can be removed by dragging them off. Double-clicking on a point will allow for direct entry of the Time and Amplitude for that point.

## FILTER

Nearly any type of filter can be created by drawing the desired frequency response curve in the graph. For more precision, double click on a graph node to directly enter it's coordinates. The actual frequency response will follow as closely as possible to that drawn. The window function used will affect the actual frequency response of the filter. The filter can be though of as an FIR filter with up to 8,192 individually adjustable parameters.
You may filter out undesired frequencies, or just keep certain desired frequencies by using the **Passive** mode. Use the **Logarithmic** mode to boost or dampen frequency components.

When the **Lock** is not set, you can choose both an Initial and a Final filter. Filtering will gradually go from the initial state to the final depending on the Transition settings.

**Graph Values** to the right of the display are given in dB or percentage (depending on logarithmic or passive mode). The values set the upper and lower limits of the graph. They are independent of the shape of the graph, which means that loading a preset will not change these two values.

The **Precision Factor** determines how accurate you want the filtering over time. A low factor, means the filter settings will change roughly, or in chunks, from the initial to the final settings. With higher factors, the filter's transitions are much smoother. In any case, the higher the precision factor, the longer it will take to filter your selection, but the nicer it may sound. The FFT (Fast Fourier Transform) function takes a large group of samples, and filters them all at once. The precision factor determines how many samples from the entire group are actually saved in the final product. A factor of two means that half the samples are saved back. A factor of 10 means that 1/10 of the samples are saved back. Since there can be only one filter setting for the entire group of samples, you will want smaller groups of samples if the settings are varying widely over short periods of time.

The **Points** parameter specifies the size of the FFT to use.  For cleaner sounding filters, use higher values.  The maximum value currently is 8192, and the value must be a power of two.  Recommend values are 1024, 2048, 4096, and 8192.

The **Windowing Function** determines the method used when filtering. DIfferent windowing methods give different frequency responses characteristics. The Hamming and Blackman windows give great overall results. The type of window determines the amount of transition width and ripple cancellation, and are in order from smallest width and greatest ripples (Triangular) to widest width and least ripples (Blackman-Harris). The filters with the least ripples are also those that more precisely follow the drawn graph, and have the steepest slopes, even though they are wider, and pass more frequencies in a band-pass operation. Try different Windows if you are not getting the effect you desire.

If **Morph** is checked, the transition from the initial filter settings to the final filter settings will actually "morph" from one to the other. If this is not checked, the settings simply change linearly over time, which means if you have a spike at 10K for the initial filter, and a spike at 1K for the final filter, the spike at 10K will gradually decrease, and the spike at 1K will gradually increase over time. If morphing is on, then the spike will "ooze" from 10K down to 1K, passing many of the frequencies in between.

Really nice effects can be heard by simply choosing the Passive mode, and having an initial setting with first half of the filter at 100%, and the second half at zero for the initial filter, and the right 1/10th or so at 100% with the rest at zero for the final filter. This selects high frequencies for the initial configuration, and low frequencies for the final configuration. To get a nice blending from high to low, choose morph to blend the two together by including all the frequency combinations between the two filters. To see exactly what is happening as the filtering changes from the initial configuration to the final, choose Transition to view the actual settings that will be used over the duration of your selection.

Various sample **Presets** are given to illustrate the various types of filters possible. You should experiment with drawing filter response graphs yourself and save your favorites into the presets.

For best results in filtering use 16-bit samples whenever possible. If your source is 8-bit, you can use the Edit:Convert Sample Type function to convert it to 16-bit. For more accuracy in using any transformation, always use 16-bit samples. If your board only supports 8-bit samples, you can work with 16-bit samples anyway, and have them converted on the fly to 8-bit at playback time by choosing the "convert 16 to 8 bit" mode in Options:Settings.

To keep the transitions from one filtered block to the next smooth, a percentage of the previously filtered block is mixed with the block currently being filtered. This will help ensure that there is no added high frequency noise when using very narrow filters. If you encounter any undesireable side effects, increase the Precision Factor to 32, 48, or even 64.

## FLANGE

What is flanging you ask? Just try it out and see! The term is coined from the flanging mechanism on the old style tape recorders which, when fiddled with, would slow down the playing of the tape, and speed it back up again when desired. That is how they got those funky psychedelic sounding recordings in the 60's. Here's how you can do it today.

**Original-Delayed Slide** This slide decides at what proportions to mix the original and flanged signal. If the Original is at 100%, no flanging is heard. If the Delayed is at 100%, a cute wavering (like a bad tape player) sound is heard. Portions of both signals need to be present for there to be cancelling out, and reinforcing of wave patterns between the two signals.

| | |
|---|---|
| **Initial Delay** | Flanging will start with the delayed signal this many milliseconds behind the original. |
| **Final Delay** | Flanging will end with the delayed signal this many milliseconds behind the original. If the delays are the same, the effect disappears, since the delayed signal will not change. |
| **Stereo Phasing** | The right channel can be at a separate delay than the left channel. A phasing of 180 will put the right channel at the initial delay value when the left channel is at the final delay value, and vica versa. |
| **Rate setting** | The **Frequency**, **Period**, and **Cycles** settings are all interrelated, and refer to the rate at which the delay cycles between the initial delay and the final delay. The flanging will cycle frequency times per second, or period seconds per complete cycle, or a total of cycles complete cycles over the entire selection. Various effects can be heard by using different settings. For example, if 0.5 cycles is chosen, the selection will start with the initial delay, and end with the final delay. If a frequency of 4 is chosen, the flanging will cycle from the initial delay to the final delay and back again 4 times per second. |
| **Invert** | Invert the delayed signal when flanging, which causes the waves to cancel out periodically, instead of reinforcing. If the mixing is at 50/50 then whenever the delay is at zero, the waves will cancel out to silence. |
| **Special EFX** | A mixture of both normal and inverted flanging, with the delayed signal summed, and a future signal subtracted out. So this option will mix not only a delayed signal, but a future one as well. |
| **Sinusoidal** | If checked, the transition from initial delay to final delay and back will follow a sine curve. Otherwise, the transition is linear, and delays from the initial setting to the final setting at a constant rate. With sinusoidal checked, the signal is at the initial and final delays more often than it is at delay inbetween. |
| **Presets** | The flange presets are pretty self explanitory. You basically have to try them out to hear the exact effects. By trying different combinations of Invert, Special EFX, and Sinusoidal, you should be able to create just the effect you want. These three options give a lot of control over the flanging effect, so experiment with them all! When you find ones you like, you can add your favorites to the list of presets. |

## INVERT

For mono waveforms, the wave is inverted (that is, crests become valleys, and valleys become crests). For stereo waveforms, this button becomes the Channel Mixer button.

## MUSIC

Now you can put your clippings to music, or just harmonize a wave using a particular chord. To choose a clipping for your sample, select the range you wish to use as a quarter note. If no range is selected, the clipboard data will be used. Note that the clipboard data will be filled with your sample automatically once music is generated, thus selecting music a second time with nothing highlighted will automatically use your last sample.

Note: This is by no means a complete full featured MIDI studio. I have provided the most basic functionality to build quick little tunes easily and on the fly.

To build a song, simply drag the notes and rests you desire to the music bar above. To sharpen or flatten a note, drag the sharp (#) or flat(b) symbol on top of the note you wish to transpose. You can move notes up or down after they have been placed, or pick them up to insert in a new position. To remove a note, pick it up and drop it off away from the bar.

Use the scroll bar to work on individual portions of the song at a time. You can scroll to write a piece as long as 256 notes.

| | |
|---|---|
| **Tempo** | The tempo is given in quarter notes (beats) per minute. Your sample's length is the length of a quarter note. If your note is longer than the period determined by the tempo, then the notes will overlap. |
| **Key** | You may choose to have your music interpreted in any of the standard key signatures. The key of C is the standard (white keys). |
| **Constant Duration** | If chosen, all notes will be the same length as the original sample, regardless of pitch. The operation that does this takes longer to calculate, but high pitched notes will be the same length as lower pitched notes. The Interval Overlap method is used with an overlap of 80% and an interval of 30 Hz. If not checked, the note is created by directly stretching or compressing the original sample, resulting in higher pitches being shorter than lower pitches. |
| **Exact Tune** | Choose Exact Tune to tune your sample so that when played at A (above middle C), the frequency of your sample is at 440Hz. If this is not checked, your sample's original frequency will be played at A (above middle C). |
| **Chords** | The triplets of numbers to the right is the chord selection box. You can choose to make a chord out of 2, 3, or 4 notes, then choose the chord from the list. Finally, pick up a chord object (the 3 notes on top of each other) and drop it on a note above. The note you drop it on will be the starting note of the chord, and the other notes will automatically appear above it in the right ratios. |
| **Clearing Chords, Sharps, or Flats** | If you want to clear a sharp, flat, or chord from a note, use the faded looking quarter-note object, and drop it on the note you wish to bring back to normal. |
| **Saving Your Songs** | If you make a cool song you want to keep, give it a name in the **Song Title** box. In the future, you can choose your song from the list of song titles that you created. The actual song data is saved in the file SONGS.INI in your Windows directory. |
| **Listen** | If you have MIDI play capabilities, you can listen to a preview of your song before actually creating it. Play begins at the leftmost note visible on the staff, which means play begins at the position you are scrolled to, and continues on to the end of the song. The music is played through channels 1 and 13 for Extended and Base level compatibility. The instrument can be chosen by typing its instrument number to the |

left. You can record music played by the listen preview button. Simply hit the record button first, then go into the music dialog and press Listen. When the song is done, hit Cancel, and then Stop to stop the recorder.

**Pink** What the heck is this button for? Well, it automatically plays the chosen instrument through the MIDI, using pink noise as the source for randomness. Maybe once out of 1 million tries, it may actually write a cool song? Tempo, Octave, and Key all affect the play of "pink" music. The only purpose for this button I have found so far, is to open the Play List first, then open the music dialog. While listening to a relaxing soundscape from the play list, you can listen to relaxing random music at the same time. (?) (!)

## NOISE REDUCTION

Background noise and general broad band noise can be dramatically reduced with minimal reduction in signal quality. The amount of noise reduction depends upon the type of background noise, and the allowable loss in the quality of the signal that is to be kept. In general, increases in Signal to Noise ratios of 5dB to 20dB can be achieved (noise is reduced 21dB and signal 1dB for example).

Two steps are required to remove noise:

**(1)** The noise level must be set so the filter knows what type of "noise" to remove. To do this, highlight a section of the waveform that has no important signal in it, and only has background noise, then press **Set Noise Level**. The statistical information about the background noise is then gathered, and you are set to remove all noise of this type from your waveform.

**(2)** Once the statistical data is gathered, highlight the section you want to remove the noise from, and choose the level of reduction you desire. A level of zero will remove the least amount of noise, and nearly no signal loss will occur. Typically the noise will be reduced about 3dB at this level. A level of 100 will remove the maximum amount of noise, lowering the noise level by about 20dB. If the signal you are trying to keep gets too distorted at this level, use lower values until a balance is reached between noise reduction and allowable signal distortion. Values any higher than 100 will guarantee loss of the signal that you want to retain.

Distortion effects may manifest themselves a "hollow" or "underwater/burbly" sounding signal, dull sounding impacts, "rolly" high end, or a "computerish" mechanical sound. These effects, if heard at all, will fall off if the noise reduction level is reduced. The amount and type distortion depends on the type of noise that is being filtered.

Besides reducing the noise level, the type of noise that is present after reduction is entirely different than the type of noise beforehand. For example, if you are trying to get rid some "tape hiss" from a waveform, the tape hiss sound will completely disappear, and in it's place about 15dB quieter will be completely different type of noise. This noise will contain all frequencies in different combinations, thus it cannot be reduced much further without noticeable signal loss. The new noise has a "burbly" or "bubbly" quality to it, and if amplified, sounds very harmonic--like those 1960's computers in old science fiction films. Since this is so much quieter than the
original noise though, it is very acceptable.

Great effects can be generated by setting the noise level to some valid signal component in the waveform, and not the background noise. Whatever frequencies are present in the highlighted selection when **Set Noise Level** is chosen will be removed when the reduction level is set to 100.

Use this function to remove tape hiss, microphone background noise, 60 cycle hum, or any noise that is constant throughout the duration of your waveform. You can even reudce the noise incurred by the sound board's circuitry during recording--just record a second of silence before whatever you want to record and tell the noise reducer to remove the sound of that silence for another 10dB dynamic range.

**FFT Size** This parameter causes the most drastic changes in quality. Good settings for this size are 2048, 4096 and 8192. This determines the number of frequency bands to analyze. Higher values will create background noise that sounds more spread out, like an orchestra tuning up. Lower values will create more bubbly effects - many short bursts of tones at different frequencies.

**FFT Precision Factor** This affects distortions in amplitude. The size above is divided by this factor to give the size
of a single chunk of audio data that is worked on at one time. Values of about 5 work great. Higher values do not make much difference in the quality of the noise reduction. If you notice a periodic wavering in amplitude, then increase this parameter to reduce the unwanted warbling or amplitude variations.

**Number of** This parameter sets the number of snapshots of noise that will be used when gathering

**Samples** statistical data. The larger this number, the more accurate the statistical data will be. A value of 64 is plenty high enough for most all applications. You will notice that when using very small numbers of statistical samples that there will be very eratic noise reduction results. The higher the number of samples taken, the more stable the noise reduction will be. For example, with more snapshots taken, a noise reduction level of 100 will most likely cut out more noise, but also cut out more original signal too. But also, with more snapshots, a low noise reduction level will cut out noise that is less lilkely to *not* disrupt the intended signal.

| | |
|---|---|
| **Save Profile** | Once the noise level is set, you can save the noise profile in a *.fft file. This file will contain information on sample type, fft size, and three sets of fft coefficients, one for the lowest amount of noise found, one for the highest amount, and one for the power average. |
| **Load Profile** | Loads any previously saved noise profile. You can load any *.fft file that Cool Edit has saved. Eventually the spectrum analyzer and Filter functions will support *.fft files. A noise profile is only compatible if it is being used on a sample of the same type when the profile was saved. In other words, a 44K stereo 8-bit sample is not compatible with a 22K mono 16-bit profile. Also, since noise profiles are so specific to the recording environment of waveform in question, even if the sample types are compatible, a profile for one type of noise will not work on another type. Even if the audio samples were recorded with the same microphone, if the recording environment is different, the type of background noise could be different. |

**Special Notes**
Noise reduction works best on 16-bit samples, although it will work perfectly on 8-bit samples. Because of the nature of 8-bit audio, it is impossible to get the noise level to less than about -45dB if even that. Noise at -45dB is very audible, as owners of 8-bit sound cards can attest. Converting to 16-bit first, then reducing the noise will produce a sample with much less noise than can be done in 8-bit alone.

The noise reduction works best if the original signal is centered. To center a signal, highlight it and choose "Center Wave" from the Amplify function. Centering the wave adjusts the DC offset to zero. If the wave is not centered, audible clicking may be heard in really quiet situations. Since centering takes out all frequencies below about 16Hz, it is completely safe to do without any ill side effects.

## NORMALIZE

Normalize will amplify the highlighted selection so it is the desired percentage of maximum. Optionally, a DC Bias can be set. Setting a DC Bias to zero will ensure that the waveform is centered on the zero voltage line.

Use this normalize function if you are recording a script in which you want to normalize a waveform to a specific percentage of maximum.

## QUICK FILTER

The 8-band quick filter allows one to customize to suit most filtering needs. The "equalizer" works pretty much the same as a standard audio equalizer does. Except that the bands are not the same as you would expect. The highest frequency band will increase or decrease the high end, but it will also increase frequencies all the way down to the lowest as well, but it will increase the high frequencies more than the low ones. The effect is close to an equalizer, but not quite. Basically, this is a handy function for changing the tone of your sample (such as noise) to make it more pleasing to the ears.

| | |
|---|---|
| **Equalizer Bars** | Adjusting these increases or decreases the frequency component specified beneath the bar. |
| **Volume Bars** | The final volume after equalizing may be adjusted to suit your needs. Checking the Lock Vol checkbox will lock the left and right scroll bars. |

| | |
|---|---|
| **Lock** | When locked, the entire selected range is equalized with the setting shown. If unchecked, the initial and final equalization settings may be adjusted, so the selection can smoothly glide from the initial equalization setting to the final setting over the range selected. |
| **Initial** and **Final** | When Lock is unchecked, you may view the Initial or Final quick filter settings. The filtering will glide from the initial settings to the final ones over the duration of the highlighted sample. |
| **Lock Vol** | When checked, the left and right volume sliders will behave as one, making the left gain the same as the right. |
| **Presets** | **Flat** - Sets all frequency sliders to zero.<br>**Bass Boost** - Increases the lowest frequencies.<br>**Loudness** - Increases low and high frequencies, and tries to leave midrange untouched.<br>**Old Time Radio** - Sounds like an old fashioned radio with a boost in the midrange and less bass and high end.<br>**Treble Boost** - Increases high frequencies only. |

## REVERB

Create high quality reverb effects. Use this function when you are looking to reproduce the effects of a certain environment - everything from your coat closet to a grand amphitheatre. Unlike echo, which generates specific echoes at specific times, the reverb function creates a very much spread out, random phase trailing of the original audio and no specific echoes can be heard at any particular time. The effect is very warm and natural. To simulate specific rooms that have echoes and reverb, use the Echo function first to get the 'size' of the room sound, then use Reverb to make it sound more natural. This function is ideal for converting Mono audio to sound as if it is Stereo. Converting a Mono sample to stereo where both the left and right channels are identical should be used as the source, then by adding some reverb, even as little as 300ms, will open up the sound so it is perceived of as being in true stereo.

**Total Reverb Length**
This is the total length of the reverberation. The signal will trail off and finally cut out at about -96dB after this amount of time. Values below 400 produce a small room environment. Values between 400 and 800 simulate medium sized rooms, and values above 800 simulate concert halls up to giant amphitheatres at delays around 3000 ms.

**Attack Time**
The amount of time it takes for the reverb to gain full strength is known as the attack time. For smaller reverb lengths, the attack time should be smaller. In general, a value of about 10% the total reverb length works well. But interesting effects can be gained by using longer attack times with shorter reverb lengths for very subtle reverb. Or, very short attack times can be coupled with long reverb lengths for other special effects.

**High Frequency Absorption Time**
In natural environments, higher frequencies are attenuated more than lower frequencies. Using this parameter, the exact time it takes for the highest frequencies to be completely cut out can be chosen. Faster Absorption times simulate rooms that are occupied and have furniture and carpeting like night clubs or theatres. Slower times (especially over 1000ms) simulate more empty rooms, like gymnasiums and empty auditoriums, where higher frequency reflections can be heard.

**Perception / Timbre**
This is another parameter to help give subtle qualities to the environment making it sound more realistic. It can be thought of as changing the width of the room and adjusting other room irregularities. With lower values, the reverb is smoother without as many distinct 'echoes'. Higher values cause more variation in the reverb amplitudes and add more spaciousness to the reverb by creating distinct reflections over time. In general, higher values (up to 60%) can be used for simulating large rooms, and lower values (down to 0%) for small rooms. But these are only suggestions. Interesting canyon effects can be created by setting this value to 100, and using a total reverb length of 2000 or more.

**Mixing - Original Signal**
This is the amount to mix the original signal into the final result. If you are trying to achieve some special effects with reverb, you may want to reduce the volume of the original signal. Or, if the reverb is so great that audio begins to clip, reduce both the original signal and the reverb mixing strength. In general, the more reverb you add, the lower the original signal volume should be. In most cases, a value of 90% or so should be fine.

**Mixing - Reverb**
This is the amount to mix the reverberated signal into the final result. A value of 100% is most natural, but you may wish to decrease this for a reverb that exists more in the background, or increase to simulate being far away from the audio source where only the reverb can be heard in greater strength than the original audio.

**Combine L&R**
In general, this should be checked for more realistic reverb, and faster calculation times. When checked, the left and right channels in a stereo source are combined before reverb is performed. This should especially be checked if you know that both channels are identical, otherwise it is just a waste of computer time. When Combine L&R is not checked, separate stereo reverb is calculated for each channel individually. The original signal will remain in the respective channels, but the reverb will carry through to both channels equally. With stereo audio where there is different information in the left and right channels, this box should not be checked. The stereo audio will be dramatically enhanced by the reverb, and sound fuller and more rich in most instances. Also, when this box is not checked, calculations take exactly twice as long to compute, since separate reverb is being calculated for the left and right channels before being recombined to the final stereo output.

The Reverb function can require large amounts of RAM while performing longer length reverbs. If not enough memory is available, the reverb will not be performed, and you should choose a shorter reverb length.

I have built a few presets to get you started. But for best results, experiment with the different parameters and I'm sure you'll find just the reverb you are looking for. The "Large Occupied Hall" gives a very nice live theatre atmosphere. The "Concert Hall Light" setting gives a nice professional performance reverb, enhancing a non-reverberated vocal singing track quite nicely.

**Presets**

**Concert Hall Light** - This gives a light reverb, which will make a mono vocal track (converted to stereo) sound very natural, as if singing in a concert hall.
**Large Empty Hall** - This sort of sounds like the gymnasium, with many distinct echoes and high frequencies bouncing around.
**Large Occupied Hall -** Same delay as the empty hall, but the *feeling* is as if the room is carpeted and filled with listeners due to absorbing more high frequencies.
**Last Row Seats** - This is a special effect, where you hear mainly a lot of reverb.
**Medium Auditorium** - Simulates a medium sized auditorium (500 seats).

**Medium Empty Room** - Simulates an empty room in your house, or the basement.
**Shower** - Very realistic Shower effect - makes vocals sound better, as if the singer sang in the shower.
**Warm Room** - Sounds like a room in your house.

## Reverse

This function simply reverses the selected audio.

One fun thing to try is to record yourself saying a short phrase, then to reverse it. Listen to how the phrase sounds in reverse, and record yourself trying to say the reversed phrase. Reverse this back to see how close you came to saying the phrase backwards. I used to do this in high school with an old reel to reel tape deck, and had a lot of fun just experimenting like this.

## Silence

This function simply forces the entire highlighted selection to silence.

## Stretch

You can choose the initial stretching percent (less than 100% will compress the wave), and the final stretching percent. By having two separate values for start and end, the wave can be stretched linearly from one ratio to another. This gives the effect of slowing down and speeding up, or raising and lowering pitch. You can alternately enter the desired final length (only accurate if both initial and final settings are set to the same desired length).

**Preserve Pitch** Lower percentages will slow down the tempo, while higher ones will increase the tempo. The pitch remains the same throughout.

**Preserve Tempo** The tempo or speed of play will remain the same. The only thing affected will be the pitch. Higher percentages will lower the pitch, and lower percentages will increase the pitch. Try using differing initial and final percentages to raise and lower the pitch without affecting the tempo. First the selection is adjusted, preserving the pitch, then the selection is squeezed or expanded, with no preservation.

**Preserve None** The tempo will slow, while at the same time the pitch will lower if percentages above 100 are used. For lower percentages, the tempo will speed up and the pitch will increase.

**Transpose** When the number of desired keys up or down are chosen, the numerical values for transposing musically are entered into the stretch sliders. For example, to make your sound as if it were the next key higher (if played on a keyboard, and black keys included) choose 1# for 1 sharp. The 'b' values will flatten or lower your sound.

**Interval Overlap**
When preserving pitch or tempo, the waveform must be elongated or truncated smoothly and preserve as much of the original information without adding noticable distortion. This is one of two methods that can be used to achieve this. The amount of distortion introduced is not dependent on the type of sample (eg music or speech). When stretching or compressing, the appropriate chunk from the original wave is output to the transformed wave, and overlapped with the previously transformed chunk.

**Interval Rate** This determines the size of a 'chunk' of audio data. Interval rates will become an audible *hollow* sound when large rates (above 50Hz) are used. If the rate is too low, echoing will be very noticable when raising pitch, or slowing down tempo, or chopped sylables will be noticable when lowering pitch, or speeding up tempo. Values of 20Hz to 40Hz usually produce good results.

**Overlapping** This determines how much of the previous chunk is overlapped with the current chunk. The overlapping can produce a *chorus* effect. To reduce the chorus effect, lower the overlapping percentage. When the overlapping is reduced, a *choppiness* to the sound may appear. Adjust the overlapping to your taste to strike a balance between chopiness and chorusing.

**Fractional Interval Overlap**
This second method of preserving pitch or tempo uses a more straightforward algorithm which guarantees the timing to be as close as the interval rate allows. This usually causes more of a reverb/echo effect with the audio, but in cases where stretching is being done at rates lower than 60 percent, or higher than 175 percent, this method can produce more desireable results..

**Interval Rate** The interval rate determines the size of the chunk of audio data in the final stretched wave. If there are any artifacts, they will be heard at this frequency. For example, an interval rate of 60Hz will cause a 60Hz distortion artifact. Values of 15 down to 7 or so seem to work best. The higher the value, the more precise the placement of stretched audio over time, but also the artifacts are more noticeable as rates go up.

**Overlapping** The overlapping determines how much the current chunk overlaps with the previous and next chunks. The maximum overlapping allowed can be as great as 1000%, in which up to 10 sections of the wave are overlapped together.

If using either of the Interval Overlap methods, you can improve the quality of the stretched audio of mono tonal (pure tone) samples by choosing an Interval Rate that is evenly divisible into the frequency of the sample. Use the Frequency Analysis window to find the sample's base frequency, then divide by an integer to get the Interval Rate. For example, if the tone was reported to be 438Hz, dividing by 20 gives 21.9Hz, so using 21.9Hz for the Interval Rate will greatly improve the quality by reducing the phasing artifacts. For very non-tonal or noisy samples, the Interval Rate does not matter as much.

**Zero Splicing**
This is the second method of preserving pitch or tempo. Less distortion is noticed when using samples that were derived from one source, and contain a relatively low number of fundamental frequencies. For example, a single instrument, or speech will work OK, while an orchestra or music will not work as well. When stretching or compressing, the wave is broken up into chunks that begin and end when the waveform crosses zero, or the midpoint.

**Cutoff Frequency** Chunks are repeated, or thrown out depending on the compression ratio and the cutoff frequency. Chunks smaller than the cutoff frequency will not be thrown out or repeated. Try cutoff frequencies between 50Hz and 300Hz for best results using this method.

**Presets** **Raise Pitch** - Adjust the settings so that the pitch of the sample will be raised by 1/4 octave.
**Lower Pitch** - Adjust the settings so that the pitch will be lowered by 1/4 octave.
**Speed Up** - Speed up the sample, so talking will be faster, but the pitch will be unaffected.
**Slow Down** - Slows down the sample while preserving pitch.

**Note:** An alternative method of stretching while preserving pitch or tempo is to do an FFT, and modify the amplitudes and phases of each frequency in the frequency domain, and then do the inverse FFT to translate back. Multiple radix FFTs are most effective. This method would be very complex, and be very very much slower than the methods currently being used. And in the end, there is very little difference in the quality of samples modified this way over just using the fast Interval Overlap method of pitch changing.

## WAVE

Audio may be Waved to produce files, that when listened to with stereo headphones, will put the listener into any desired state of awareness. For example, by listening to waved files, you can easily achieve states such as deep sleep, theta meditation, or alpha relaxation. Because of the nature of this function, it only works on **Stereo** waveform data, and to be effective, must be listened to with stereo headphones. The Wave function spatially locates the audio left and right, in a circular pattern over time. In order to spatially encode the signal, either the left or right channel is delayed so that the sounds will appear at each ear at different times, tricking the brain into thinking they are coming from either side. When this is done at frequencies of 3Hz and above, the brain will start synchronizing at the same frequency, increasing its output of Delta, Theta, Alpha, or Beta frequencies.

Since this function periodicaly spatially locates audio left and right, it can also be used to make mono stereo audio (where left and right channels are the same) sound as if they are coming from the left, then the right, then the left again at the chosen frequency.

The Frequency, Intensity, and Centering can all change dynamically based on the graph. The graph represents time along the horizontal. Points at the top of the graph correspond to the High Settings, while points at the bottom correspond to the Low Settings. This allows you to vary the frequency encoded, or the intensity at which the brainwave signal is encoded, or the perceptual location of the sync signal over time, or all three of these at once.

**Frequency** This is the actual frequency that audio is waved back and forth at that the brain responds and synchronizes to. See the section on creating brainwave files for more information on what specific frequencies do.

**Frequency Graph** Click on the graph to add new control points. Drag a control point up or down, or off the screen to remove. Choose the highest and lowest frequencies that are represented on the graph with the scroll bars. You can also choose the highest and lowest Intensities and Centering values. Gliding about 4 to 5 Hz over 2 minutes works nicely. If large variations are done in short time spans, the effects are not as pronounced. For example, after 5 minutes of Theta waves, if 30 seconds of alpha waves are generated, and returned to theta, the listener will become slightly awake, and aware of his surroundings for that brief moment. The effect is like all of the sudden changing gears, and you stop thinking about whatever it was you were thinking about, and become aware that you were thinking about it, but aren't any more.

**Intensity** This is the intensity of the brainwave encoding. Higher intensities work well with lower brainwave frequencies. Beta waves should have intensities below 25 or so, while Delta waves work better with intensities above 60.

**Centering** You may choose to have your brain think the synchronization frequencies are coming from the left or right. This may affect the left or right hemispheres more intensely, but that's only a guess. Mixing a file that has been waved to the left with one that has been waved to the right (in the same frequency range within 2 Hz) has interesting effects.

**Musical Source** If the selection being waved is musical, checking this will calculate the wave patterns in such a way as to eliminate clicks and pops. If the source is noisy (waterfall, ocean, nature recordings, etc.) do NOT check this. If you do, it will actually add interference. Since noise is based on "randomness", the clicks and pops are inaudible.

**Smooth Wave** When checked, the actual audio appearing at the left and right channels is smoothed out, but the spatial encoding is identical. The left and right channels will delay and un-delay following a smooth curve such that the delay difference between the left and right channels follows a sine wave, and the brain will hear the audio travelling around the head in a circe. When Smooth Wave is not checked, the net delays are the same, but are achieved by holding one channel constant (at no delay) while the other channel is delayed following half a sine wave. Then the other channel is delayed while the first is held constant. The boundary between holding constant and delaying is discontinuous in that the dD/dt (difference in delay over time) jumps from zero to a positive delay value without hitting any values inbetween. When Smooth Wave is checked, the dD/dt is always continuous. This will also cause less noticeable distortion in either channel when heard independently.

For special spatial panning effects, choose wave frequencies of 1Hz or less. A mono source (left and right the same) will appear to move from left to right and back at period of 1/frequency. For example, a frequency of 0.1Hz will pan the audio in a "full circle" over the period of 10 seconds.

## Waveform Generation Functions

Cool Edit can generate a variety of audio signals from scratch. These signals can be inserted into the current waveform, or generated as a waveform in its own right. Some very interesting effects can be created by experimenting with the various generation functions. The following cover the items available from the **Generate** menu.

### GENERATE NOISE

Generate random noise in a variety of colors. Each color has its own characteristics. One use for generating noise is to create a waterfall-like sound which is ideal for use as a carrier wave for brainwave synchronization files (see the Wave function). It is also great for making weird effects by Flanging and Filtering (with filter morph).

**Color**

Noise can be a variety of colors, which describe its spectral composition.

| | |
|---|---|
| **Brown** | Brown noise has a spectral frequency of 1/f^2. Which means, in English, that there is much more low-end, low-frequency components to the noise, which results in thunder and waterfall like sounds. Brown noise is called that because, when viewed, the wave follows a Brownian motion curve. That is, the next sample in the waveform is equal to the previous sample, plus a small random amount. This gives the appearance of a mountain range when graphed. The wave pattern is very predictable. |
| **Pink** | Pink noise has a spectral frequency of 1/f and is found mostly in nature. It is the most natural sounding of the noises. By equalizing, rainfall, waterfalls, wind, rushing river, and other natural sounds can be generated. Pink noise is exactly between brown and white noise (which is why some people used to call it tan noise, but pink was more appealing). It is neither random, nor predictable. It has a fractal like nature when viewed. When zoomed in, the pattern looks identical to when zoomed out, except at a lower amplitude. |
| **White** | White noise has a spectral frequency of 1. In other words, equal proportions of all frequencies are present. Because the human ear is more succeptable to high frequencies, it sounds very "hissy". White noise is generated by choosing random values for each sample. |

**Style**

Noise can be generated in a variety of styles for your listening pleasure.

| | |
|---|---|
| **Spatial Stereo** | This is noise generated by using 3 unique noise sources, and spatially encoding them to appear as if one is coming from the left, the other from the center, and the last from the right. When listened to with stereo headphones, the mind perceives sound coming from all around, not just in the center. To choose the distance from center of the left and right noise sources, you can enter a delay value in microsoeconds. About 900 to 1000 microseconds corresponds to the maximum delay perceivable, and a delay of zero is identical to Mono noise (left and right channels are the same). |
| **Independent Channels** | This type of noise is generated by using 2 unique noise sources, one for each channel. The left channel's noise is completely independent of the right channel's noise. |
| **Mono** | Mono noise is generated by using 1 noise source, with the left and right channels set equal to the same noise source. |

| | |
|---|---|
| **Inverse** | Inverse noise is generated by using 1 noise source as well, but this time with the left channel's noise exactly inverse of the right channel's noise. When listened to with stereo headphones, the effect is that of the sound coming from the center of the listener's head instead of out in space somewhere. |

**<u>Other Options</u>**

| | |
|---|---|
| **Intensity** | With higher intensities, the noise becomes more erratic, and sounds harsher and louder. |
| **Duration** | This is the number of seconds of noise to generate. If long periods of noise are desired, it is faster to generate a short period of noise (about 10 to 20 seconds), Delete excess noise at the beginning and ending of the noise so that the waves are starting and ending at the midpoint, Copy, then Loop Paste (see Paste Special) as many times as needed. |

## GENERATE SILENCE

Insert any amount of silence. This function is most useful in extending the length of the audio signal before performing an echo.

## GENERATE TONES

Generating tones is a great way to provide a base sound to create spectacular special sound effects.

| | |
|---|---|
| **Base Frequency** | The main frequency (F) that will be used for sound generation. |
| **Modulate By** | Enter the variation in frequency you wish to hear. For example, choosing 100 will oscillate the tone being generated between 50 minus and 50 plus the base frequency. |
| **Modulation Frequency** | This is the rate (times per second) at which the frequency modulates. Entering a value of 10, for example, will generate tones that warble at the rate of 10 times per second. |
| **Flavor** | Choose the type of waveform to use. Sine waves sound soft, while Triangle and Sawtooth waves are sharper. Each flavor has a particular sound unique unto itself. |
| **Modulate over Source** | If a selection was highlighted, it will be modulated by the tones based on the normal tones settings. Instead of generating new tones, the currently highlighted wave data will be "ring modulated" by multiplying the tone by the data underneath. This is great for adding really weird special effects. |
| **Start Phase Difference** | The left channel can be out of phase with the right channel. If you wish this, choose the amount of phase shift here. A value of 180 will be completely out of phase. |
| **Phase Change Rate** | The phase difference can vary over time at the rate specified. For example, a phase change rate of 2 Hz will cycle the Phase Difference two full cycles per second. |
| **Duration** | This is how many seconds of tones you wish to produce. |

| | |
|---|---|
| **Frequency Components** | You can choose up to 4 overtones, and mix them at any proportion. Adding overtones gives truer musical effects, such as piano and pipe organ. The frequency multiplier associated with the overtone component is fully adjustable. A value of 10, for example, means that the associated slider will control the amount of *10 x Base Frequency* there will be in the final tone. These multipliers can vary over time by choosing different initial and final settings for them. |
| **Lock** | If checked, the overtones, base frequency, modulation, modulation frequency and frequency multipliers are constant -- they do not vary over time. Uncheck this box to dynamically change the proportion of any overtone over time by choosing the inital and final proportions. You can also dynamically change the base frequency, modulation, modulation frequency, and frequency multipliers for interesting effects. |

| | |
|---|---|
| **Initial**, **Final** | You can choose to adjust the initial or final overtone proportion settings when the overtones vary over time. You can also choose different base frequencies, modulations, and modulation frequencies. To create a rising tone, for example, choose a low initial base frequency and a high final base frequency. Other interesting effects can be obtained by having different *modulation frequency* and *modulate by* settings. |
| **Flip** | Check (or uncheck) Flip to swap the initial and final settings. This is very useful if you wish to generate another tone *after* the last tone that was just generated. Whatever overtone settings the last tone had as the final settings can be used as the initial settings on this tone by clicking Flip. All you need to do is click Final, and enter new final overtone values. This way the tones will flow together smoothly. |
| **Volume** | The Volume sliders are used to select how intense the tones will be in each channel. Both channels can be controlled independently when generating stereo tones. |

This function supports Presets so you can save your favorite combinations.

Experiment with all the settings for various wild effects.

## Options

Many aspects of the program can be tailored to your specific needs. Also, many tools are available to make your wave editing a breeze. The following cover the items available from the **Options** menu.

### Loop/Play Toggle

Audio selections can be played a single time through, or played as a continuous loop. Choose the Loop/Play mode before playing the audio (changing the mode while audio is playing has no effect). You can also use the Loop mode to test how a signal will sound if looped on a sampling keyboard before the excess is trimmed.

### Monitor Source

The audio record in channel can be monitored to make it easier to set proper recording levels. In stereo mode, the top red bar is Left and bottom is Right. A peak level indicator will stick on the display for a short period after the peak. When finding a good recording level, it is best if the red bar never goes completely to the right, since this indicates that the audio will be clipped when recorded. Try to ensure that the meter stays on average near the middle of the display, and no more than about three quarters of the way up for loud transcients. For most accurate results, choose File:New and choose a sample rate before monitoring.

### Settings

These settings affect the entire application in one way or another, and the values are stored in the [Cool Edit] section of WIN.INI.

**Disk Icon Interpretation** The Disk icon can be interpreted as:

- Save Now (saves what you are working on under the same filename seen in the title bar without asking if you wish to overwrite the original) or
- Save As (Always brings up the Save As dialog box to enter the title to save the file as)

**Viewing Mode** When zoomed in closely to a wave, individual samples can appear as dots, or as a continuous line.

**Play/Record Buffer Size** The number of seconds to reserve memory for recording and playback. Increasing this will allow more multitasking while audio is being played, but it takes more memory. If this value is too small, there may be too much choppiness in your recordings and playbacks. If your recordings are getting all "chopped up", or you cannot Stop after you've started recording, increase the buffer size, or switch to a faster hard drive (Use a non-compressed hard drive for example).

**Temp Directory** This is the location that Cool Edit will use to save all of the temporary files, which includes the disk image of the file currently being edited. It is best to choose a large, fast hard drive for this. If you no for certain that the drive is not compressed, then do not check the **Compressed** box, otherwise check this box. If you have trouble recording high quality audio for more than a minute (or even less), then make sure the Compressed Drive box is not checked. When checked, DOS is asked for the number of bytes remaining on the hard drive since this value does not change linearly with time on compressed drives.

| | |
|---|---|
| **Undo Directory** | This is the location that Cool Edit will use to save all the Undo files. Up to 3 undo files will be present per instance of Cool Edit being run, to accommodate the trhee-level undo. This directory can be different that that of the Temp Directory. It makes no difference whether the drive use for Undo is compressed or not - but a non-compressed drive will still be faster. |
| **Highlight After Paste** | After doing any Paste operation (this includes Paste Special), you can have the inserted selection automatically highlighted, or just have the cursor at the end of the pasted selection. Not highlighting after pasting makes it easier to do multiple pastes one after the other. |
| **Play 16-bit files as 8-bit** | If your sound board is only capable of 8-bit audio, you can still create and edit 16-bit audio files. When you choose Play, the audio data will be converted to 8-bit before being sent to the sound board. |
| **Enable Undo** | If this is checked, the undo function is enabled and running. After making a change, you can choose Edit -> Undo to back up one step. You may want to disable the undo feature if you know you are not going to undo something. For example, if you are running a function on a 5 minute file, you may not want to wait while the undo information is saved. Up to 3 levels of undo are saved. |
| **NT Display Compatibility** | If this is checked, then the program "locks" while the green waveform is being drawn. With Windows NT, other apps are not paused when this is going on, and the extra code needed to "multitask" with other apps does not need to run. If you are running Windows NT, and this is not checked, you may experience various screen redraw problems. |
| **Smooth 16-bit to 8-bit Conversions** | If checked, when 16-bit audio is pasted into an 8-bit waveform, or 16-bit audio is Opened As 8-bit, the conversion is smoother sounding, retaining more of the original signal but is a tad slower. |
| **Enable Toolbar Help** | When the mouse is over a toolbar button for more than a few hundred milliseconds, a small help box appears to describe the button's function. Anyway, if it is annoying for you, you can turn it off here. |
| **Play from cursor** | When no selection is highlighted, audio can be either played from the current cursor location to the end of the view, or always from the left edge of the view to the end of the view. |
| **Maximum Display on Load** | This is the maximum number of seconds of audio to display when a file is first loaded. When working with large files, you may wish to limit the initial display area to 10 or 20 seconds so you don't have to wait for the entire waveform to draw. Setting this value to zero means there is no limit on the initial display size. |
| **Time Code Display** | Double-clicking on the time boxes will change their display format. When in Hours:Minutes:Seconds:Frames format, you can customize the number of frames per second that will be displayed by entering the number of frames per second here. |

| | |
|---|---|
| **Spectrum Analyzer Resolution** | This is the number of bits of accuracy to display the spectral analysis when Edit->Spectral View is chosen. Each bit will take approximatly twice as long to draw, but the resolution will be more precise. The number of data points displayed is equal to 1/2 of two to the value given. In other words, a value of 7 will display 64 points along the y axis, a value of 8 will display 128, 9 for 256, 10 for 512 and so on. On most displays, a value of 10 ias high as you will ever need to go. |

## INFO

Extra information can be included in your .WAV files using the RIFF LIST INFO and DISP type 1 formats. This information should (depending on future wave editors) stay with your sound through it's lifetime. Other wave editors should preserve all of the fields you see here, but experience has shown that some editors only keep partial information.

Be sure to put proper information in it's place!

| | |
|---|---|
| **Display Title** | This should describe the sound, or text (if there are words in the wave). This field should be as short as possible, since it will be displayed in OLE objects and the like. |
| **Icon** | Any DIB or BMP file can be inserted, but preferable a 32 X 32 16-color would be best. The Media browser uses this size to display a picture representing the sound. Other OLE compatible applications can use the above display title, and/or the bitmap to represent your waveform. |
| **Original Artist** | The one who created the sound initially. Examples are: Beatles, Pat Sejek, Fred Flinstone |
| **Name** | The title of the wave. This is your chance to put a name with your audio "artwork". Examples are: Thunderstorm At Night, Forest Stream |
| **Genre** | The Genre of the original work. With audio, let's try things like musical classifications, etc. Examples are: Cartoon Voice, New Age, Instrument |
| **Key Words** | In the future, sounds may be searched for by key words. Please separate key words by a semicolon followed by a space. For Example: Violin; Hayden; Johann Strauss |
| **Digitization Source** | Where was the sound digitized from. A tape deck, CD, or maybe directly from a microphone? Maybe describe the board used here too, like Sound Blaster Pro, or MediaVision. For Example: DDD CD to MediaVision Pro 16 |
| **Original Medium** | Where did the sound come from originally. Examples: Live Band, Flute, Moog, Voice |
| **Engineers** | Store the name(s) of the engineer(s) who worked on the file, or edited the file. Please separate names by a semicolon and a space. When a new person edits the file, they can add their name to the list. For example: John Cravitz; Fred Millstone |
| **Digitizer** | Who is the technition that did the actual digitizing? They should put their name right here. |
| **Comments** | This is for making any comments you wish. Feel free to include any special effects or enhancements you made to any preexisting waves so that the editing history can be tracked. Spec says you should not hit <Enter>, but let the typing wrap around on its own. End each sentence with a period. For Example: It took me 12 hours to get this recording right. Ted added echoing effects using Cool Edit. |

| | |
|---|---|
| **Subject** | This Describes the content of the file. Feel free to include a description of the instruments used, where someone can find the song recorded, etc. Line returns are OK, and are created by pressing Ctrl+J. Sometimes copyright information is placed here as well. For Example: The shakuhachi of Japan.<Enter><Enter>The shakuhachi was developed in the 15th century from a Chinese end-blown flute, called the chiba. |
| **Source Supplier** | The name of the person, or organization who supplied the original source material. Let's use this field for the names of record companies, or whoever supplied you with the source. Examples: MCA Records, Ann Wilson (if recorded live) |
| **Copyright** | Any copyright information for this file should go here. Example: (c)1992 G. Willikers Corporation. All rights reserved. |
| **Software Package*** | The software used to digitize and edit this file. Cool Edit will place its name in this field unless marked not to. |
| **Creation Date*** | The date that the subject matter was created. The date should be in the format yyyy-mm-dd, using '0' as a place holder in single digit values. For example, if the date the original recording was made was July 30, 1988 then it would be written as: 1988-06-30. Cool Edit will fill this field with the current date if it is empty and not marked to skip this operation. |
| **Fill * fields automatically** | If checked, the Software Package and Creation Date fields are filled automatically by Cool Edit. If you are saving .WAV files that should not have any extra information embedded in them, then check this field to prevent Cool Edit from adding these fields. Some older software may not follow the .WAV file specification properly, and as such, may not be able to play files if extra information is attached to them. |

## FREQUENCY ANALYSIS

The Frequency Analysis window displays the current spectral content centered about the cursor. Moving the mouse point over the analysis display will reveal the frequency and amplitude (amount of the frequency in question) of the signal at the cursor. As long as this window is open, it will automatically update if the cursor position changes. If a selection is highlighted, the frequency plot represents the center of the highlight instead of the cursor.

To view variations easier, the **Normalize** option is provided. If checked, the most intense frequency is plotted at the top of the display, with all others relative to it. If viewing in spectral mode, the frequency analysis window displays a close-up view of the vertical slice at the cursor. The bright spots on the spectral view plot will show up as high points on the frequency analysis graph.

The fundamental frequency is also displayed below the frequency plot. Cool Edit tries its best to find the fundamental by analyzing the locations of all the peaks on the graph. Some times, depending on how pure the signal is, the value could be off. To make sure, test a few points along the note you are analyzing to see if the same frequency is found.

Because a large number of custom colors are used for this display, the look of a highlighted selection on 256-color displays may look strange. The colors seen when nothing is highlighted though should always range from dark blue (little signal) through lighter blue, purple, red, orange and finally yellow (most signal).

## CD Player

If you have the [MCI] CD Audio driver loaded, you can control the CD player with a standard set of control icons. You can also name your CD, and the individual songs as well for display the next time you use the same CD.

| | |
|---|---|
| **Tracks List** | Click on any track number to start playing that track. |
| **Time Readout** | Displays the current time in minutes:seconds into the current track. |
| **Title Display** | Displays the title of the CD. The title defaults to the length of the CD. You can type in the proper title of the CD here. If a track is currently being played (if the track number is highlighted in the track list), then the title reflects the title of the current song. As you are entering song titles, you may use the TAB key to jump to the next song to easily enter the titles for the entire CD. Titles are saved in the file COOL.INI. |

Currently, only up to 64K of titles data can be stored (the limit for any INI file), so be aware that you may not be able to enter in data for every CD you own. It may be better, if you have tons of CDs, to just enter in the title of the CD, and only enter in the song titles of those you listen to the most. If there is a great need to allow more than 64K of titles data, I will work around the limitation to allow it...

| | | |
|---|---|---|
| ■ | **Stop** | Stops CD playing. Play will resume at the start of the CD. |
| ▮▮▮ | **Pause** | Pauses the CD. Play will resume at the same location. This button will turn into a Play button when pressed so that play can be resumed by pressing it. |
| ▶ | **Play** | Starts the CD either at the beginning of the disk, or at the paused location. This button will turn into a Pause button when pressed, so that play can be paused by pressing it. |
| ▶▶ | **Scan Back** | Rewinds the CD 10 seconds. |
| ◀◀ | **Scan Forward** | Forwards the CD 10 seconds. |
| X | **Mark** | Mark the location currently being played. |
| →X | **Goto Mark** | Go to the location that was marked earlier. |
| ⏏ | **Eject** | Spit out the CD if that is possible on your player. This will also Insert it. |

## Cue List

A cue list is a list of time offsets into the wave file. A cue can be either a point, specifying a cursor position, or a range, specifying a selection. You can easily jump to a cue position in a wave by double-clicking on the position in the list, or selecting the cue position, and pressing Goto. Cue ranges can later be arranged in a play list to be played back in any order, with a specific number of loops if desired. A maximum of 96 cues may be entered. At any time, even while Playing a file, the cues may be added to the cue list. Use F8 as a shortcut to add the cursor location or highlighted range to the cue list.

| | |
|---|---|
| **Add** | Add the currently highlighted selection, or cursor position to the cue list. Items will be displayed in temporal order, with the earliest cue position at the top of the list. |
| **Remove** | Remove the selected cue position from the list. |
| **Label** | Short text label describing the selection. |
| **Description** | A textual description of the wave data if necessary. Also can be used as a comment. |
| **Goto** | Goto the selected cue position, or highlight the selected range. Double clicking a cue item acts as if Goto were pressed for that item. |
| **Merge** | Merge will create a cue range that spans the two cue items selected (whether they are ranges or markers themselves). To select more than one cue item in the list, hold down on the CTRL key when selecting, or click and drag over more than one cue item. The name used for the new merged item will be the same as the earliest item chosen in time (the highest item in the list). The information typed into the Name and Description fields for the second item being merged will be lost. |

**<u>Markers</u>**
The cue list can be used anytime to mark your current selection so you can return to it later. If you would like Cool Edit to remember your highlighted selection, or just your current cursor point, click Add in the cue list, and quickly type a name for your selection. In the future, if you want to return the cursor to that point, or re-highlight that selection, double-click the name or choose the name and click Goto.

One great use for markers is to highlight a wave from the zero crossings.
- Go to the start of wave portion you wish to highlight, and zoom in as far as needed to position the cursor exactly on the zero-crossing point.
- Add that position to the cue list.
- Now zoom out, go to the end of the wave portion, and once again zoom in to find the ending zero crossing.
- Add this new position to the cue list.
- Select both start and end points in the cue list (hold down on the CTRL key to select more than one), and press Merge.
- Voila! Double-clicking on this new range selects the range with the endpoints that were just merged. You can choose "Zoom In" now to see your entire selected wave portion if you like.

**<u>Assigning Cue Ranges To Keys</u>**
If you wish to assign any cue range you have added to a key on the keyboard, give the cue range a label of the form KEY N, where N is any key on the keyboard (capital letters only). When you go back to editing the waveform, pressing the key will play the cue range you selected. You can assign any portion of the waveform to any key on the keyboard this way.

The cue list is saved in the .WAV file format in the 'cue ' chunk. Additional information about the cue position, like label, description, and length of sample, are placed in the 'adtl' list in the 'labl', 'note', and 'ltxt' chunks.

## PLAY LIST

he play list is a listing of cue ranges that can be played in any order, and looped a specified number of times. The play list is used in conjunction with the cue list. Maximum size of a play list is 64 entries.

| | |
|---|---|
| **Add Before** | Add the currently highlighted selection from the cue list to the play list. The selection is inserted before the currently highlighted play list item, or at the end if nothing is selected. |
| **Remove** | Remove the selected play list item from the list. |
| **Loops** | The number of loops to loop the selected cue range in the play list. |
| **Play** | Play the cue ranges in the order listed, looping selections if necessary. Play begins at the currently highlighted item in the play list, or the entire list is played if [end] is selected, or there is no selection. |
| **Autocue** | Play the currently highlighted item in the play list (or the first item if nothing is highlighted), looping if necessary, and stop on the next item in the play list. Thus, every time Autocue is pressed, the next item in the play list is played. |

The play list is saved in the .WAV file format in the 'plst' chunk.

## SCRIPTS (A.K.A. MACROS)

Scripts are similiar to Macros. Your exact mouse moves, and tweaking of parameters is not stored, only the final result when you click "OK". Clicking "Cancel" in a dialog while recording a script does nothing, as if you never brought up the dialog to begin with. The Undo action *is* recorded into a script. If you use Undo while recording a script, the Undo function must be enabled when that script is being played.

Multiple scripts can be kept in one script file, and identified by name. At any time, you can edit the script file directly to take out steps, rename scripts, remove unwanted scripts, etc.

There are various types of scripts, which depend on when you initiated the recording:

- Scripts that were recorded with File->New as the first action, and must be played back on a blank, empty waveform.
- Scripts that were recorded while a waveform was open, but not highlighted anywhere. During playback, actions begin at the insertion point in the waveform, and may affect any part of the entire wave if present.
- Scripts that were recorded while a portion of a waveform was highlighted. All actions in the script pertain only to the portion that is highlighted, leaving the rest of the waveform untouched.

Scripts that run during all of the above conditions will be displayed, but only the ones recorded under the same circumstances will be allowed to run. In other words, if a script recording started when a portion of a wave was highlighted, then you will only be able to run that Script when something is highlighted.

Scripts are very useful for remembering how you generated a particular sound effect. Use the script to reproduce the sound effect without having to save the entire waveform. This is especially useful when generating large brainwave "theta" files, which can take monsterous amounts of space. By generating the file once, with the scripting turned on (record), you can generate the file again at any time in the future, and save all that hard drive space. You can also pass along scripts to your friends across email or BBS
systems, since they take nearly no memory to store.

When running a script, you can either stop at each dialog box, or have the script automatically run through completion by using the **Stop at Dialogs** checkbox. Stopping at each dialog box is handy if you wish to 'tweak' the parameters while the script is running. You may also choose to be notified when the script is done by checking **Alert When Complete**. A dialog box notifying you of the completion will be displayed when the script is finished.

After recording a script, you may enter a **description** at the bottom of the dialog to go with the script you just recorded. This description will appear when the user of the script highlightes the script to run. Note: the only time you can edit the description is after recording, not before, and not after it has been added to a script collection file. To get around this, you can still edit the description at any time by pressing the Edit button to edit the text file directly.

To record a script, enter the name of the script in the space provided, and press **Record**. When you are done, come back to the **Cool Scripts** dialog and press **Stop**. Enter a description, then press **Add** to add the script to the currently opened collection. Before adding, you can open another collection or create a new one by pressing **Open/New.**

A single script can be run on a batch of files by pressing the **Batch Run** button. For more information, see the section on **Batch Processing** next.

**Pause at Dialogs** At each dialog, the script will stop to allow you to modify the values to the function. Pressing Cancel at this point will stop the script, pressing OK will continue it.

**Alert when complete** When the script is finished, a dialog box will signal the completion of the script if this option is checked.

**Execute Relative to Cursor** When running a script that was recorded when a waveform was loaded but there was no highlight, it can be run by playing back all the operations relative to the beginning of the file or to the beginning of the cursor. For example, the Sound Effects scripts require you open a waveform (it can be blank) first. Checking this option will insert the effect at the cursor, otherwise the effect will be inserted at the start of the file.

Important Note: Other buttons and functions are not disabled while the script is running. Therefore, do not use the other functions until the script has stopped playing, or you have Canceled out of the script by pressing **Cancel** at one of the dialog boxes.

**<u>FXNS2.SCP Sample Collection</u>**

**Description** These are five sample functions I came up with. Cross Fading is useful if you are going to loop the sample. The last portion of the sample is overlapped with the first portion, and the amount of overlap is different for each Cross Fade script. Full cross fading fades the last half of the sample with the first half. Soft cross fading fades the last 5% with the first 5%, and hard cross fading fades the first 0.4% with the last 0.4%. Make Piano Keys will take the highlighted sample and stretch and compress them to vary the pitch, making 13 copies of the original, each at a different pitch. Each pitch is assigned to a key on the keyboard through the cue list. This turns your keyboard into a simple sample player. Reverse Echo is just that -- the echo function, but the echoes go in reverse.

**How To Use** These sample functions work on a highlighted selection. Open a waveform, and highlight the portion you wish to operate on, then run the script.

**SNDEFX2.SCP Sample Collection**

**Description** Here are some nifty sound effects, and a small song (very small) I wrote. If you run the Cool Song script, you can then go to the Music function, and enter a name for the song to save it under. This script generates a short note, and then uses the Music function to build the song. The other four effects are just weird effects using the tones or noise functions with other transformations. If you've watched "Dr. Who", you may recognize the Cool Lasers sound effect.

**How To Use** These sound effects work in a currently opened waveform. Open a New (blank) waveform in any sample rate setting you desire for the quality you would like, and run a script. These sound effects can also be inserted into an existing waveform, and will be inserted just as if the Paste command were used.

**MINDSNC2.SCP Sample Collection**

**Description** Included are four "Tones" synchronization scripts, which have a binaural beat pattern (two differing tones in each ear) overlaid with the corresponding "Waved" pink noise. Choose Loop Play and listen to the audio as long as you like. Each Tone script stimulates a different brainwave frequency, from Delta to Theta to Alpha, and an "Earth" tone of 7.83Hz. The "Music" scripts have "Waved" music overlaid with the pink noise for a relaxing effect. The Creativity Theta Session is similiar to the sample theta session described with the Wave function, and lasts 1/2 hour. The session starts at Alpha, goes down to Theta, and stays there with a few bursts into Alpha and back.

**How To Use** Open a new blank waveform of any Stereo sample setting you wish. I suggest using at minimum 22K 16-bit stereo, but the synchronization effects will still work at lower sample rates and 8-bit. Once you have a blank waveform to work with, run one of the scripts.

## BATCH PROCESSING

A single script can be run repeatedly over a group of source files. The script must have been recorded in a “Works on Current Wave” mode, that is, before the script was recorded there must have been an open waveform (perhaps blank) and no highlighted selection. The **Batch Run** button will only be selectable if a script of this type is selected from the Scripts list.

Any number of **Source Files** can be chosen as long as they are in the same directory (sorry about the limitation). Press the **Browse** button to choose these wave files. The wave files can also all be in different formats if desired.

After each file has had the script run on it, it will be saved to the **Destination Directory**. You can choose the **Output File Format** that all the waves will be saved as, as well as enter the appropriate options for the file format if the format supports options.

File names can be modified slightly before being saved. The filename extension will change to that of the file format being saved automatically (eg *.AIF). If another filename extension is desired, or some modification to the filename portion is desired, the filename template can be modified. Use the question mar '?' to signify that a character does not change, and a '*' to denote the entire original file name or entire original file extension. Here are some examples of how filenames will be saved given the original file name and the filename template:

| Original Filename | Template | Saved As |
|---|---|---|
| zippy.aif | *.wav | zippy.wav |
| toads.pcm | q*.voc | qtoads.voc |

|  |  |  |
|---|---|---|
| funny.out | b???????.* | bunny.out |
| biglong.wav | ????.wav | bigl.wav |
| bart.wav | *x.wav | bartx.wav |

Choosing **Overwrite existing files** will always perform the script on the file in question and save it to the destination, even if a file of the same name already exists at the destination. If this box is unchecked, and the destination filename already exists, the batch will not even attempt to run the script on the file in question, but skip it instead.

In most cases, you can check the **Disable Undo** option to disable the Undo function during the batch run. Unless the batch was written expecting the Undo function to be enabled, this is a completely save thing to do, and it speeds up processing because undo information does not continually need to be saved.

If a source file is unreadable, or in a RAW type format without any header information, then the batch needs to know what file format to assume for the data. This ensures that the batch will run continuously without interruption by dialogs asking for input data formats on head-less data.

## WAVE DEVICE SELECTION

If you have multiple sound cards, or multiple output devices (such as a sound card and the PC speaker), choose the input and output devices you wish to use by selecting **Select Wave Device** from the **Options** menu. Different devices may have differing capabilities, which will be shown in the Capabilities display as each device is chosen.

If your system is equipped with MIDI devices, you may also choose the MIDI IN, and MIDI OUT sources. Currently MIDI IN is not used for anything, and MIDI OUT is only used in the Music function for previewing songs.

These settings are remembered in the [cool edit] section of your WIN.INI, which means if you install a new sound driver or card, Cool will not access it until you choose it from this dialog first.

It is possible for two separate instances of Cool to be playing at the same time, if each one has a different waveform output device selected!

## Questions and Answers

**Q: I had saved some presets using a previous version of Cool Edit, and now they're gone. Where did they go?**

**A:** They are most likely still in your COOL.INI file, but in creating new versions, sometimes it is impossible to keep the parameter orders and such the same for some functions. Look in your COOL.INI file and find your previous presets and print them out or write them down. Perhaps you can re-enter them from looking at the data. Also, you can try to just copy the entries from the previous function section to the new one. Sections that are 100% compatible are [Channel] from 1.33 and the new [Channel Mixer] section; [NewFlanger] from 1.33 and the new [Flanger2] section. Entries from these can be copied straight across. Entries from 1.33's [Filter] section can be copied straight across to the new [Filter2] section, but be aware that any filters designed are going to be assumed as if they were done at 44.1khz. This means a bandpass filter at 5khz done on a 22khz wave will now be assumed to be a 10khz filter at ALL samplerates. The new [Amplify2], [Stretch2], and [Tones2] sections are not compatable with the previous 1.33's [Amplify], [Stretch], and [Tones]. Sorry for the inconvenience.

**Q: I had written a script using Cool Edit 1.33, and now it won't run properly. Help!**

**A:** Sorry about that. In making modifications and everything more generalized, some incompatibilities may have arisin. When making the conversions, I tried to favor keeping Scripts running properly over remembering presets properly. You can print out the SCP file containing your invalid scripts and then try to reconstruct it by recording a new script and following along what you had done previously.

**Q: I cannot load or save .WAV files properly. When I try to load a .WAV file, I have to enter the sample rate, and there is garbage noise at the start and end of the sample. What's going on ?**

**A:** You may have another application running with the same name as the file filter you are having problems with. For example, if you cannot load normal Windows PCM .WAV files (which uses the wave.flt file filter), then there may be another application running that is named wave.exe. If this is the case, just rename the .flt file so the 8-character names are different. In the case of wave.flt, rename it to coolwave.flt.

**Q: Cool Edit doesn't seem to remember the last file type I opened, and keeps wanting me to open files in a different file format, even though I never use that format. Help?**

**A:** The problem may be the same as mentioned above: Another application is running with the name of the file filter you are using to open the file.

**Q: I get an error message when I try to record. What's up?**

**A:** Check the capabilities of your sound card. You may be trying to record at a rate not supported by your hardware. Some older boards can play at higher rates than they can record. You may also be trying to record in 16-bit mode for an 8-bit card. When Cool Edit is started, the playback abilities are displayed in the audio format box (but this does not necessarily mean you can record at the same rates).

**Q: I get an error message when I try to play a wave. How come?**

**A:** Your board probably does not support the audio data sample rate, or sample size. Try checking "Play 16-bit files as 8-bit" in the Settings box. This will convert any 16-bit audio data to 8-bit as the file is played. If you still get an error, your board does not support the 8-bit version as well. This could happen if you are trying to play a stereo 44.1K 16-bit waveform, and your board does not even support 44.1K 8-bit stereo. The documentation that came with your sound board will give the maximum sampling rates it can play. Also, Cool will detect the maximum playback rate on startup and display it in the middle status window.

**Q: I just installed a 16-bit audio card, but my 16-bit sound files still sound awful. Should I take my sound card back?**

**A:** No. Your card is probably fine. Check to see that the "Play 16-bit files as 8-bit" box is *not* checked in the Settings dialog. If it is checked, your files are being converted to 8-bit before being played. Also be sure you are using the right DMA settings. The lower DMA channels can only support 8-bit audio. Please check your sound board manuals for this information.

**Q: Why does it take forever to do things like Filter, and to use Spectral View?**

**A:** These functions use the Fast Fourier Transform (FFT) to convert the waveform from temporal data to frequency data. The FFT does "zillions" of floating point operations to accomplish this. If you do *not* have a co-processor (this includes 486SX users!) then these operations are going to take a very long time. A co-processor will speed these operations up by ***at least*** a factor of 10, sometimes 20!

**Q: Why does it take so long to save as ADPCM Wave format?**

**A:** ADPCM is a compression scheme, which compresses 16-bit samples to 4-bits while retaining much of the quality at higher sample rates. If the multiple pass option is chosen, which provides the highest quality, each block of wave data is compressed seven different ways, and the way which sounds most like the original signal is then saved.

**Q: I am using Windows NT. Why doesn't my screen redraw when I make a change?**

**A:** Normally, Cool Edit is set up to "multitask" when the waveform is being drawn, so as not to lock up the PC when drawing large waves. NT automatically multitasks, which throws things out of sync. Make sure the NT Compatibility option is checked under Settings if you experience screen redraw problems.

**Q: How long did it take you to make this program?**

**A:** It took me over a year and a half of programming in the evenings and weekends to produce what you see here. I had actually started writing some of the waveform transformation functions six months prior when I wrote a DOS only version of Cool, which only worked with Sound Blaster files. I first ported that functionality to Windows, then added on functions and features from there, and eventually ended up with this creation. The brainwave function (Transform:Wave) was written around April to May of 1992, and eventually released to ShareWare in July of 1992. I experimented for several months with different sound/relaxation techniques before finding the one which worked extremely well.

**Q: Why are some functions not selectable?**

**A:** If you have the unregistered version of Cool, you chose which functions you wanted to use for the editing session. The others will not be selectable until you choose them on the next editing session. Also, some functions only work on stereo files, such as Wave and Channel Mixer (which becomes Invert for mono waves).

**Q: How can I see my wave size information in samples instead of time?**

**A:** Double-clicking on the time (or samples Start and End) window will toggle the display between time and samples. This, and double-clicking on the wave to select all, are the only functions that do not have a corresponding menu item or button associated with them. Other shortcuts are double-clicking on the waveform type display to change the waveform interpretation (ie interpret the 44K wave as a 22K wave), and double-clicking on the green bar to bring up the viewing samples data entry box.

**Q: Why do my recordings sound choppy?**

**A:** You probably have too small of a buffer size in the Settings dialog. A minimal buffer size is about 4 seconds. Using a compressed hard drive on a slow PC could also eat up so many CPU cycles, that there isn't time left to do recording. Either try adjusting the buffer size up or down, or record at a lower data rate (ie 8 bits instead of 16, or 32K instead of 44K). I tried very hard to ensure that recordings would sound perfect, without any data loss. If you have problems, see if other recording software has the same problems. If so, you may have a hardware incompatibility between you sound card and your main board, video board, or other installed boards.

**Q: My computer crashed while Cool Edit was running, and now I lost alot of hard drive space. What happened?**

**A:** Cool Edit uses temporary files for the editing of waves, and the undo buffer. Since some of these files may have been open (the editing temporary buffer is always open) during the crash, the files were never closed properly. Exit Windows, and type {\b CHKDSK /f C:} (use whatever drive letter is missing the space). If you get "Convert lost chains to files?" or something like that, answer yes. The missing data will now be in the root directory as *.CHK files, which you can then safely delete.

**Q: I keep getting "parity error" messages, and the program keeps crashing. What's going on?**

**A:** Some older style PCs do not handle memory properly, and Cool just makes the problem orse since it is very CPU intensive. In some cases the problem is that using the CPU so much heats it up to the point that it fails. Sometimes adding another fan inside the PC eliminates the problem. The "parity error" message appears when one of the bits hat were saved into DRAM have changed unexpectedly (indicating bad memory). Most boards have the first 64K of memory on the motherboard, so the problem could be either in your memory arrays, or on the motherboard.

**Q: Cool does something really weird that I didn't expect, and I think it's not working right. What's a person to do?**

**A:** A person should send me mail explaining the bug they've found, and I will respond by either finding a fix in the program, or just letting them know that the program is supposed to do that.

# Brainwave Synchronization

## <u>About Brainwave Files</u>

The wave option works like many meditation tapes and light/sound devices on the market, which range in price from $200 to $500. There are even boards available with plug in glasses (which have blinking lights) for your PC in the price range of $495. I think the files created using the 'Wave' transformation are even more powerful, and are definitely more pleasing to the ears. Most other devices and tapes have a "humming" sound or some other tones to induce the right brainwave frequencies. This program allows you to use ANY sound to encode the frequencies with. The most effective I have found are by using the Noise Generator, which creates pleasing waterfall like sounds. This function only works on **stereo** waveforms, and the effects work if only if listened to with **stereo headphones**.

Listening to sounds that have been waved for periods of 5 minutes or more will produce the desired state of awareness in the listener. Sessions of 25 minutes or so work really well! After being "sync'd in" for 10 minutes or so, changes in patterns can be programmed more quickly, and the brain will respond accordingly.

**<u>Major brainwave pattern frequencies and possible uses for brainwave synchronization</u>**

| | | |
|---|---|---|
| **Delta** | 1-3 hz | Deep sleep, lucid dreaming, increased immune functions. |
| **Theta** | 4-7 hz | Deep relaxation, meditation, increased memory and focus. |
| **Alpha** | 8-12 hz | Light relaxation, "superlearning", positive thinking. |
| **Beta** | 13-25 hz | Normal state of alertness, stress and anxiety. |
| **Gamma** | 30 hz on up | Hyper-awareness??? |
| **High Gamma** | 200+ hz | Not sure exactly what these do... |

**Immediate Relaxation and Stress Relief** - Choose between 5hz and 10 hz for different levels of relaxation.
Meditation - Choose between 4hz and 7hz, either cycle between a few, or stay at a particular frequency for different results.
**Sleep Replacement** - A 30 minute session at 5Hz replaces about 2-3 hours of sleep, allowing one to wake up in the morning more refreshed. Try listening 1/2 hour before waking up in the morning, or 1/2 hour before going to bed.
**Improved Sleeping Patterns** - Any of the Alpha and Theta frequencies (8Hz to 4Hz) for 30-45 minute sessions at the same time each day.
**Treatment of Insomnia** - Choose between 4hz and 6hz for starters (the first 10 minutes), then go into frequencies below 3.5hz (for 20-30 minutes), settling on about 2.5hz before fading out.
**Improved and Lasting Sense of Well Being** - Try Theta (4Hz to 7Hz) for 45 minutes, daily.
**Creative Visualization** - About 6hz for a while, then up to 10hz works well while using visualization techniques.
**Alleviation of Migraines and Headaches** - Experiment with Alpha and Theta combinations. Try and visualize the pain getting smaller and smaller until it disappears.
**Reduction of Depression Symptoms** - Again, Alpha and Theta combinations, mostly theta(?)
**Self Hypnosis** - Choose about 8hz to 10hz while playing any self-hypnosis tape, or guided meditation.
**Accelerated Learning** - Choose about 7hz to 9hz while playing any learning tapes, like foreign language tapes, etc. to increase comprehension. Also, while studying, take breaks every half hour and listen to 10 minutes of Alpha (10Hz) while reflecting on the material you just learned.
**Subliminal Programming** - Choose 5hz to 7hz while playing your favorite subliminal tapes, or make your own by recording some affirmations, and mix pasting (Edit:Paste Special) them from the clipboard at barely audible volumes.
**Improve Intuition (or ESP?)** - Theta frequencies help in this area, 4hz to 7hz.
**Reaching Higher States of Consciousness** - Theta again, with daily half hour minimum sessions. Give at least a month for results.
**Quick Refresher on long days** - Low Alpha 8hz to 10hz for about 15 minutes works well. Sort of induces a cat-nap.
**Increased Immune System** - Relaxing to Alpha and Theta combinations daily. Learning how to relax, and relaxing more often can lower blood pressure and increase the body's natural defenses. Using Alpha Synchronization (8Hz to 12Hz), expect similar increases in the neuro-chemical levels of Norepinephrin (11%), Serotonin (21%) and Beta-Endorphins (25%).

**DISCLAIMER**

By using this program, you agree that the author (David Johnston) will not be responsible for any damage as a result, direct or indirect, of using this program. The author makes no claims about the effectiveness of these sounds for any particular purpose. The user is encouraged to do his/her own research into the area of brainwave synchronization via auditory stimulation.

**WARNING**
Sounds generated by the wave function may not suitable for epileptics or persons undergoing psychiatric treatment.

## <u>About Carrier Waves</u>

A carrier wave is needed to transport the brainwave frequencies. Because the carrier wave is not what you hear through the headphones directly, you do **not** need to buy super high-end headphones (5Hz-25KHz) to reproduce the effects. These sounds may be recorded using any stereo cassette recorder and played back on any stereo cassette player without losing effectiveness. In other words, your headphones do not need to be able to reproduce a 5Hz signal if you are generating a 5Hz theta-frequency brainwave file, and your tape deck does not need to be able to record frequencies this low either. The brain *does* however respond better to the lower frequencies because of the nature of the synchronization algorithm, so the better the headphones you buy, the more dramatic the results may be. The best headphones are the kind that cover the entire ear, so outside noise does not get in. Plus, these headphones have much higher response to low frequencies. The active ingredient, so to speak, are the frequencies from about 40Hz up to about 2khz depending on the frequency being encoded and the intensity.

Carrier waves must have some correlation between the left and right channels, no matter how slight. So mono (total correlation), inverse (total negative correlation), and spatial (natural recordings that have some of the same sounds coming in both channels) will work great.

The best sounds to use as carriers are sounds that are spread across the entire frequency range, or at least most of the lower frequency range. Good examples are ocean, waterfall (most any recordings from nature), and noise generated by this program. Experiment with mono (both left and right channels the same), inverted (like mono, but the left channel is the inverse of the right, obtained by using the Channel Mixer), and spatial stereo (spatially encoded sounds in nature, recorded with microphones about 9 inches apart to simulate separation between the ears). But don't let this stop you from digitizing your favorite music, and using it as a carrier, or converting your favorite to a mono or inverted wave.

To generate a carrier wave, you can do three things:

**Record a sample** - Once recorded, use the Channel Mixer to create a mono, or inversed wave. The channel mixer will also allow you to put in just the amount of correlation you desire (for example, a 20% mixture of both channels, leaving the rest untouched.) Or just leave it the way it was recorded. You may find changes in effectiveness of the brainwave files depending on how you use the Channel Mixer. Keep in mind that this function only operates on stereo waves, so when "mono" is mentioned, it means that the exact same signal is present on both channels--the left channel and right channel are the same.

**Generate Tones** - You may use the Generate Tones function to find a pleasing, relaxing tone for the background (but I find "noise" sounds more relaxing). The way tones work the best is if the left channel's tone frequency is 5-6 Hz different from the right channel's tone. This creates a beat pattern equal to the frequency difference, which the brain responds to somewhat (this is the property that many theta-inducers rely on). To do this, generate one tone with left volume at 40, and right volume at zero. Then generate the second tone with the left and right volumes reversed. Finally, Paste Special (with overlap) one tone on top of the other. Use low frequency tones, like 50Hz to 120Hz for best results. These tones, by themselves, will help coerce the mind into the state associated with the difference between the frequencies. For example, for a theta state of 6Hz, use a 70Hz and a 76Hz tone. Combining this tones sample with an existing brainwave file, by overlap pasting at a quiet volume (20%) is even more effective.

**Generate Noise** - Use the Generate Noise function (pink and brown work best) in any of the modes: mono, inverse, or spatial stereo (independent channels noise will **not** work as a carrier for brainwave frequencies at all, since there is no correlation between the left and right channels). I find that using pink noise in spatial stereo, and running it through the Quick Filter to get rid off some of the “edge” if any works the best. I have also found Inverse to work quite well too, but the brainwave “effect” is more pronounced, and can be distracting, and some sound boards have trouble reproducing sound that is inversed between channels.

Once you have found a pleasing sound, about 10 seconds or so of a monotonous sound (tones, river, waterfall, noise...) you're ready to start. If a monotonous sound is used, more disk space can be saved because we will use the play list to repeat portions. If a music sample were used, it is quite noticeable that the same 10-second piece is being played over and over and over again.

If you're curious you can also spatially locate a mono sound to the left or right. Do this if you wish to have the illusion that a particular sound is coming from one side or the other. The function works by pasting a mono sound sample into a stereo waveform, and using the Digital Delay function. Having a quiet "ping" (generated by using the sine wave tone generator with the bell curve envelope) play spatially on the left, then on the right at about 5 second intervals is very relaxing.

## ENCODING BRAINWAVE INFORMATION

There are two types of brainwave files that you can create: A **flat file**, and a **cued file**. The flat file takes more memory, and plays straight through from beginning to end, while the cued file is actually contains pieces of the entire audio program, that when played in the proper order become the brainwave file. The cued file takes less memory, and can very quickly be modified at any time by re-arranging the audio pieces. The average length of a cued file is about 3-4 minutes for a program that can last as long as desired. The flat file is a standard wave file, which means to create a long program, you must have enough space for it. The only advantage to using a flat file is if you are waving music, since music cannot be split into pieces and rearranged, otherwise it would sound discontinuous. Creating brainwave files using the flat file method will be discussed first, since it is more straightforward.

### Flat Brainwave File Generation

Create a file the length you wish to make your relaxation program using the carrier wave(s) of your choice. Either record music, or use the pink noise generator and copy and paste (or Paste Special) to the desired length. If you are using a monotonous sound, you would be better off using the cued file method. Lengths of good relaxation programs vary from 15 to 30 minutes, and beyond. This means you must have enough hard drive space for the entire file. Since the temporary file takes up hard drive space as well, the maximum size of file you can create, and be able to save, will be one that takes up half of the initial free hard drive space.

Use the Wave function to encode the brainwave patterns into the carrier wave by highlighting a section of the wave, or the whole thing, and choosing Transform:Wave, or click the wave icon. With the wave transformation, you have complete control over the brainwave frequency being encoded, the strength of the signal, and the positioning of the signal left or right. Over the selection highlighted, the intensity, and position remain constant, but the frequency can be varied using the graphical input control. See the section on Authoring Brainwave Files to learn what settings to use for the Wave function, and how to build effective files.

Once the entire file has been waved to your satisfaction, you can save the file if you wish, and play it using the Play button. An interesting side effect is that different sounds are heard if you listen to one channel, listen to both channels with one ear, or listen to each channel with each ear.

### Cued Brainwave File Generation

These files contain many short snippets of brainwave encodings at different frequencies. Each snippet is cued using the Cue List, and a Play List is generated by adding entries from the Cue List, and looping them if necessary. To listen to a cued brainwave file, you must use the Play button in the Play List dialog box.

First you must figure out how you want to divide up the brainwave program (your 20-30 minute masterpiece) into components. For example, you may want to have patterns of 5Hz, 7Hz, and 9Hz at different points in the program. In this case, you will need at least three pieces for your creation. The actual file will just be 10 seconds of carrier wave at 5Hz, followed by 10 seconds at 7hz, followed by 10 seconds at 9Hz. All the pieces are placed in the cue list by highlighting the piece, and choosing **Add**. It is best to add the piece to the cue list once it is created, or pasted at the end of the current waveform. To create the final program, the pieces are added to the Play List in the order you wish to listen to them. Each piece can be looped if needed. So a
20 minute program can be generated from 3 10-second pieces by adding the cues to the play list and looping.

First you need to create 10 to 20 seconds of carrier wave, and save in a special file in case you need the carrier wave again later. Highlight the wave, and Edit:Copy. When you need another copy of the initial carrier wave, you need only to Paste it.

Add the first carrier wave snippet to the Cue List by pressing the **Add** button in the Cue List dialog. Give the cue for this snippet a name that reflects the waveform transformation you will be using, for example, “6Hz to 5Hz drop”.

Use the Wave function to encode the proper patterns into the carrier wave. Look at the section on Authoring Brainwave Files to learn what settings to choose.

Click past the end of the wave file (make sure the rightmost part of the file is in view), and choose Paste to insert another copy of the carrier wave. Once you do this, you can add the newly inserted selection to the cue list, and give it a name. Repeat the step above for creating a brainwave encoding over the carrier wave you just inserted. Do this as many times as needed until you have all the pieces you need to build the final brainwave file.

Once all the pieces have been generated, add them in the order you like to the play list. To make pieces last longer (if the beginning and ending of the piece are at the same brainwave frequency), increase the number of loops for that entry in the play list.

When Played from the play list, the pieces will be played in the order shown, and looped if necessary.

To get familiar with the cue list, and play list, open one of your favorite wave files, and highlight sections then add them to the cue list. After you have a few selections in the cue list, add them to the play list, and choose a loop count of greater than one for some of them. Choose Play from the play list, and listen to what you've just created.

**Authoring Brainwave Files**
After learning about carrier waves, and encoding procedures, all you need to know is what frequencies to use, and when to use them during the course of the listening session. Once you know what frequencies to use, and at what intensity, you can generate the completed file using either of the methods above.

Effective brainwave files have some sort of encoding going on the entire length of the session. For the first 3 minutes or so of the session, the listener will not be in a "relaxed" state, and will not respond greatly to the frequencies being presented. During this *warm-up* period, gradually decreasing from about 12Hz down to 8Hz works nicely. After about 4 minutes, the listener's brainwave patterns will start to synchronize with the patterns in the headphones, and the serious brainwave programming can begin.

Frequencies of 8-10Hz correspond to an alpha state -- light relaxation, like a quick afternoon siesta. Frequencies of 6-7Hz correspond to a theta state -- meditation. 4-5Hz correspond to deep relaxation. You can create a session that is constant, in one of these states, or create a session that dynamically flows from one to the other. When going down in frequency, give the listener about one minute to *catch up*, and stay in sync with the wave. Going up in frequency does not require the listener to catch up. In other words, if you go from 6Hz down to 4Hz over a 20 second timespan, and hold at 4Hz, the listener may not be at 4Hz for another minute. When going from 4Hz to 8Hz in 20 seconds, the listener will be at 8Hz at the end of the 20 seconds. It appears to take extra time when going down in frequency, but no extra time when going up. This basically holds true for the first 20-30 minutes of a session. After that, the opposite tends to occur. It is easier to go lower than go higher. This means that to bring a listener from 4Hz (where she has been for the last 30 minutes) up to 12 Hz, it should be done over a 5 minute period or so. One nice *trick* to do is to keep the listener at around 4-5 Hz for a while, then about once every 2 minutes, go up to 8Hz and back over a 20 second span. This will *alert* the listener slightly, and make them aware for a few seconds of what they are thinking. This is great for getting creative insights and the like. It acts as a sort of *window* to the subconscious, allowing one to remember what is going on. It's kind of like remembering dreams: you do it better if you are awaken in the middle of one.

Another effective method of producing relaxation files is to overlap them. That is, have portions that are one frequency, and slightly spatially located to one side overlapped with a slightly differing frequency spatially located slightly to the other side. This gives the listener the chance to *decide* which frequency to be at, and gives them more freedom over the experience. For example, a session could go from 8Hz to 4Hz over 10 minutes overlapped with 7Hz to 5Hz over the same 10 minutes.

For nice *super-relaxing* effects, generate panning waves (frequencies of 0.05 to 0.2) over your session after encoding the initial brainwave patterns. For example, if you are generating a brainwave file out of 20-second pieces, after generating the main brainwave frequency over the 20 second period, generate a panning wave of 0.05 or 0.1 (which means a period of 20 or 10 seconds) with an intensity of about 50 or so. This will make the sound appear to shift left and right to the listener over a 20 or 10 second period. Now, overlapping a 24-second piece panned at 0.125 (8 second period) at 5Hz with a 0.167 (6 second period) at 6Hz will combine the
practices of multiple frequencies with panning for an extremely super-natural effect!

Once you get started creating a few files, and see what the different frequency ranges do, you will become familiar with the different effects and how to generate just the effects you want.

High Gamma frequencies of 200Hz or more seem to help in relaxation, and do something I'm sure, but I don't know what... yet. This is an area you can experiment with. When generating frequencies above 40Hz or so, it is best to keep the intensity very low, like 7 or 8. The higher the frequency, the lower the intensity has to be, otherwise the encoding will overwrite itself and the signal will be lost.

<u>**Sample Theta File - Step-by-Step**</u>

**1** Create a new blank file with **File:New**. Choose a **Stereo** file, either 8 or 16 bit and a 11025, 22050, or 44100 sampling rate. The final file size will be one of the following sizes listed below depending on your choice:

| | 11025 | 22050 | 44100 |
|---|---|---|---|
| 8-bit | 2.6M | 5.2M | 10.5M |
| 16-bit | 5.2M | 10.5M | 21.2M |

You must make sure you have enough memory for a file of this size, plus an additional meg for working space. If you plan on saving the file when you are done, you must have at least **twice** this amount of hard drive space available, since a temporary file is used instead of memory while working on the wave.

**2** Choose **Generate:Noise**. Select **Pink, Spatial Stereo** (500 µSeconds) for **15** seconds at an intensity of **3** . This is usually the longest portion of the generation of brainwave files. Because of this, it is advised that you save this piece of *noise* so that in generating future files, you can just load in this pre-calculated noise as a starting point.

**3** Choose **Edit:Copy**. From now on, we will paste the noise in when we need it!

**4** Make sure the noise is highlighted. If it is not, select all by double-clicking on the waveform until it is highlighted.

**5** Choose **Add** in the Cue list, and give the entry a **Label** of **10Hz to 8Hz**, and a **Description** of "**Warm-Up**"

**6** Choose **Transform:Wave** to bring up the brainwave dialog box. Enter **10** for the **Highest Frequency**, and **8** for the **Lowest Frequency**, and an **Intensity** of **35**. On the graph above, click the leftmost dot, and drag it to the top of the graph. Click the rightmost dot, and drag it to the bottom of the graph. This will product a frequency encoded at 10Hz at the beginning, and glide down to 8Hz by the end. Choose **OK** to generate the encoding. This shouldn't take nearly as long as it did to generate the noise.

**7** Click the mouse at the rightmost portion of the wave (just beyond the *black* waveform display area). When you do this, the yellow cursor arrows should be all the way to the right of the wave. You must always add new pattern blocks at the **end** of the current waveform.

**8** Choose **Edit:Paste** to insert another copy of the original noise that we had copied originally.

**9** Create the following pattern blocks as before (following the steps 5 to 8) , except with the following values for the cue list and waveform transformation:

| Label | Description | Hi Freq. | Lo Freq. | Intensity |
|---|---|---|---|---|
| *(Graph should go from left=highest to right=lowest)* | | | | |
| 8 Hz | Alpha | 8 | 8 | 37 |
| 8 to 6Hz | Glide Down | 8 | 6 | 38 |
| 6Hz | High Theta | 6 | 6 | 40 |
| 6 to 5Hz | Deeper Theta | 6 | 5 | 45 |
| 5Hz | Theta | 5 | 5 | 50 |
| *(Graph should look like an upside-down "V" for Spike)* | | | | |
| 5-8-5 | Spike | 8 | 5 | 50 |
| *(Graph should go from left=lowest to right=highest for Awake)* | | | | |
| 5 to 12Hz | Awake | 12 | 5 | 40 |

**10** Once all the blocks are generated, and in the cue list, Add the pieces to the play list by selecting the wave portion in the cue list and clicking **Add** in the play list. Select the pieces listed below in the order given. After doing so, select each item in the play list, and change the **Loops** for each so the final play list looks like this:

| | |
|---|---|
| (1) | 10 to 8Hz |
| (3) | 8Hz |
| (1) | 8 to 6Hz |
| (7) | 6Hz |
| (1) | 6 to 5Hz |
| (18) | 5Hz |
| (1) | 5-8-5 |
| (12) | 5Hz |
| (1) | 5-8-5 |
| (12) | 5Hz |
| (1) | 5-8-5 |
| (12) | 5Hz |
| (1) | 5-8-5 |
| (12) | 5Hz |
| (1) | 5 to 12Hz |

When you choose **-Play-** from the play list, the sequence will be played in the order given, looping the number of times specified. This list gives a 21 minute theta session, with bursts into alpha at four points.

**11** If you wish to save this piece, and have enough hard drive space, you can do it now. The wave is complete. Enjoy.

**<u>How to use brainwave synchronization files</u>**

Once you have created your brainwave file (15 minute files on up work best), Loop Play them for a longer listening time. Sessions of 15 minutes or more work best. It is best to listen to the sessions lying down in a quiet place where you will not be disturbed. If there is no place like this near your PC, it may be a good idea to record the session on tape and listen to it where you can be comfortable and relaxed. When you're fully
comfortable, start the session, close your eyes, and let the magical sounds from Cool Edit do the work. Remember, this only works if you listen to the sounds with stereo headphones.

You may notice helicopter, or "washing" type noises moving around in your head. These sounds are actually created inside your head, and are not coming directly out of either channel from the sound board. It is this noise that is doing the work of helping your brainwaves get synchronized to the patterns you have chosen. When I have mixed two different (but similar in frequency ranges) brainwave files together, I have noticed a jet airplane noise moving slowly from left to right in the background. Some people don't hear these
artifacts at all, while others hear them extremely well.

Another side effect is that of a wandering mind. When I use frequencies under 8hz, I find myself thinking of the strangest things. You may find that you are not thinking of anything in particular, and your thoughts become very interesting. The feeling is also "warm" and "happy" for some people. Others

start recalling their favorite memories as a child, even some they thought they had forgotten forever!\
par\par

After a session of 15 minutes or more, you may feel quite refreshed, light, airy, clear-headed, etc. I always find myself feeling very good afterwards. Some claim that doing this for 30 minutes a day can result in subtle but great changes in your life. ESP experiences increase, and you may be able to reach new levels of awareness in your everyday life.

## References on Brainwave Synchronization

Adams, H. B. (1965). A case utilizing sensory deprivation procedures. In L. P. Ullman & L. Krasner (Eds.), *Case Studies in Behavior Modification.* New York: Holt, Rinehart & Winston.

Adrian, E. D. & Yamagiwa, K. (1935). "The origin of the Berger Rhythm." *Brain*, 58, 323-351.

Atwater, F. H. (1988). "The Monroe Institute's Hemisync process: A Theoretical Perspective." Faber, Va: Monroe Institute.

Bandler, R. (1985). "Using Your Brain--For a Change." Moab, UT: Real People Press.

Barber, T. X. (1957). "Experiments in hypnosis." *Scientific American*, 196, 54-61.

Bremer, F. (1958a). "Physiology of the corpus callosum." Proceedings of the Association of Research on Nervous Disorders, 36, 424-448.

Bermer, F. (1958b). "Cerebral and cerebellar potentials." *Physiological Review*, 38, 357-388.

Brackopp, G. W. (1984). Review of research on Multi-Modal sensory stimulation with clinical implications and research proposals. Unpublished manuscript--see Hutchison (1986).

Budzynski, T. (1973). "Some applications of biofeedback-produced twilight states." In D. Shapiro, et al (Eds.), *Biofeedback and Self-Control*: 1972. Chicago: Aldine-Atherton.

Budzynski, T. H. (1976). "Biofeedback and the twilight states of consciousness." In G. E. Schwartz and D. Shapiro (Eds.), *Consciousness and Self-Regulation*, Vol. 1, New York: Plenum Press.

Budzynski, T. H. (1977). "Tuning in on the twilight zone." *Psychology Today*, August.

Budzynski, T. H. (1979). "Brain lateralization and biofeedback." In B. Shapin & T. Coly (Eds.), *Brain/Mind and Parapsychology.* New York: Parapsychology Foundation.

Budzynski, T. H. (1981). "Brain lateralization and rescripting." *Somatics*, 3, 1-10.

Budzynski, T. H. (1986). "Clinical applications of non-drug-induced states." In B. Wolman & M. Ullman (Eds.), *Handbook of States of Consciousness.* New York: Van Nostrand-Reinhold.

Budzynski, T. H. (1990) "Hemispheric asymmetry and REST." In Suefeld, P. Turner, J. W., Jr. & Fine, T. H. (Eds.), *Restricted Environmental Stimulation*, New York: Springer-Verlag.

Cade, C. M. & Coxhead, N. (1979) "The Awakened Mind: Biofeedback and the Development of Higher States of Consciousness." New York: Delacorte Press.

Cheek, D. (1976). "Short-term hypnotherapy for fragility using exploration of early life attitudes." *The American Journal of Clinical Hypnosis*, 18, 75-82.

Davidson, R. J., Ekman, P., Saron, C. D., Senulis, J. A., & Friesen, W. V. (1990). "Approach-withdrawal and cerebral asymmetry: Emotional expression and brain physiology." *Journal of Personality and Social Psychology*, 58, 330-341.

Deikman, A. (1969). "De-automatization and the mystic experience." In C. T. Tart (Ed.), *Altered States of Consciousness.* New York: John Wiley & Sons.

Deikman, A. (1971). "Bimodal consciousness." *Archives of General Psychiatry*, 25, 481-489.

Donker, D. N. J., Nijo, L., Storm Van Leeuwen, W. & Wienke, G. (1978). "Interhemispheric relationships of responses to sine wave modulated light in normal subjects and patients." *Electroencephalography and Clinical Neurophysiology*, 44, 479-489.

Evans, F. J., Gustafson, L. A., O'Connell, D. N., Orne, M. T. & Shor, R. E. (1966). "Response during sleep with intervening waking amnesia." *Science*, 152, 666-667.

Evans, F. J., Gustafson, L. A., O'Connell, D. N., Orne, M. T. & Shor, R. E. (1970). "Verbally-induced behavioral response during sleep." *Journal of Nervous and Mental Disease*, 1, 1-26.

Evans, C. & Richardson, P. H. (1988) "Improved recovery and reduced postoperative stay after therapeutic suggestions during gneeral anaesthetic." *Lancet*, 2, 491.

Felipe, A. (1965). "Attitude change during interrupted sleep." Unpublished doctoral dissertation. Yale University.

Foster, D. S. (1990) "EEG and subjective correlates of alpha frequency binaural beats stimulation combined with alpha biofeedback." Ann Arbor, MI: UMI, Order No. 9025506.

Foulkes, D. & Vogel, G. (1964). "Mental activity at sleep-onset." *Journal of Abnormal Psychology*, 70, 231-243.

Glicksohn, J. (1986). "Photic driving and altered states of consciousness: An exploratory study." *Imagination, Cognition and Personality*, 6, 167-182.

Green, E. E., Green, A. M. (1971). "On the meaning of the transpersonal: Some metaphysical perspectives." *Journal of Transpersonal Psychology*, 3, 27-46.

Green, E. E., & Green, A. M. (1986). "Biofeedback and States of Consciousness." In B. B. Wolman & M. Ullman (Eds.). *Handbook of States of Consciousness.* New York: Van Nostrand Reinhold.

Harding, G. F. & Dimitrakoudi, M. (1977). "The visual evoked potential in photosensitive epilepsy." In J. E. Desmedt (Ed.), *Visual Evoked Potentials in Man: New Developments*. Oxford: Clarendon.

Henriques, J. B. & Davidson, R. J. (1990). "Regional brain electrical asymmetries discriminate between previously depressed and healthy control subjects." *Journal of Abnormal Psychology*, 99, 22-31.

Hoovey, Z. B., Heinemann, U. & Creutzfeldt, O. D. (1972). "Inter-hemispheric 'synchrony' of alpha waves." *Electroencephalography and Clinical Neurophysiology*, 32, 337-347.

Hutchison, M. (1986). *Megabrain*. New York: Beech Tree Books. William Morrow.

Hutchison, M. (1990). "Special issue on sound/light." *Megabrain Report:* Vol 1, No. 2.

Iamblichus. "The epistle of Porphyry to the Egyptian Anebo." In *Iamblichus on the Mysteries of the Egyptians, Chaldeans, and Assyrians*. Trans. by Taylor, T. London: B. Dobell, and Reeves & Turner, 1895.

Janet, P. (1889). *L'Automatisme Psychologique.* Paris: Alcan.

Koestler, A. (1981). *The Act of Creation*. London: Pan Books.

Kooi, K. A. (1971). *Fundamentals of Electroencephalography.* New York: Harper & Row.

Kubie, L. (1943). "The use of induced hypnagogic reveries in the recovery of repressed amnesic data." *Bull. Menninger Clinic*, 7, 172-182.

Lankton, S. R., & Lankton, C. H. (1983). *The Answer Within: A Clinical Framework of Ericksonian Hypnotherapy.* New York: Bruner/Mazel.

Leman, K. & Carlson, R. (1989). *Unlocking the Secrets of Your Childhood Memories.* Nashville: Thomas Nelson.

Lilly, J. C. (1972)). *Programming and Metaprogramming in the Human Biocomputer.* New York: Julian.

Lubar, J. F. (1989). "Electroencephalographic biofeedback and neurological applications." In J. V. Basmajian (Ed.), *Biofeedback: Principles and Practice*, New York: Williams & Wilkins.

Mavromatis, A. Hypnagogia: *The Unique State of Consciousness Between Wakefulness and Sleep.* New York: Routledge & Kegan Paul, 1987.

Miller, E. E. (1987). *Software for the Mind: How to program Your Mind for Optimum Health and Performance.* Berkeley, CA: Celestial Arts.

Moscu, K. I. & Vranceanu, M. (1970). "Quelques resultats concernant l'action differentielle des mots affectogenes et nonaffectogenes pendant le somneil naturel." In M. Bertini (Ed.), *Psicofisiologia del Sonno e del Sogno.* Milan: Editrice Vita e Pensiero.

Moses, R. A. (1970). *Adler's Physiology of the Eye: Clinical Applications.* St. Louis: Mosby.

Nemiah, J. C. (1984). *The unconscious and psychopathology.* In S., & Meichenbaum, D. New York: John WIley & Sons, pp. 49-87.

Oster, G. (1973). "Auditory beats in the brain." *Scientific American*, 229, 94-102.

Peniston, E. G. & Kulkowski, P. J. (1989). "Alpha-Theta brainwave training and B-endorphin levels in alcoholics." *Alcoholism*, 13, 271-279.

Richardson, A. & McAndres, F. (1990) "The effects of photic stimulation and private self-consciousness on the complexity of visual imagination imagery." *British Journal of Psychology*, 81, 381-394.

Rossi, E. L. (1986). *The Psychobiology of Mind-Body Healing.* New York: W. W. Norton.

Rubin, F. (1968). (Ed.), *Current Research in Hypnopaedia.* London: MacDonald.

Rubin, F. (1970). "Learning and sleep." *Nature*, 226, 447.

Schacter, D. L. (1977). "EEG theta waves and psychological phenomena: A review and analysis." Psychology, 5, 47-82.

Schultz, J. & Luthe, W. (1959). *Autogenic Training: A Psychophysiological Approach in Psychotherapy.* New York: Grune & Stratton.

Sittenfeld, P., Budzynski, T. & Stoyva, J. (1976). "Differential shaping of EEG Theta rhythms." *Biofeedback and Self-Regulation*, 1, 31-45.

Stoyva, J. M. (1973), "Biofeedback techniques and the conditions for hallucinatory activity" In McGulgan, F. J. and Schoonover, R. (Eds), *The Psychophysiology of Thinking.* New York: Academic Press.

Svyandoshch, A. (1968). "The assimilation and memorization of speech during natural sleep." In F. Rubin (Ed.), *Current Research in Hypnopaedia.* London: MacDonald.

Swedenborg, E. *Rational Psychology.* Philadelphia: Swedenborg Scientific Association, 1950.

Tomarken, A. J., Davidson, R. J., & Henriques, J. B. (1990). "Resting frontal brain asymmetry predicts affective responses to films." *Journal of Personality and Social Psychology*, 59, 791-801.

Townsend, R. E. (1973). "A device for generation and presentation of modulated light stimuli." *Electroencephalography and Clinical Neurophysiology*, 34, 97-99.

Tucker, D. M. (1981). "Lateral brain function, emotion, and conceptualization." *Psychological Bulletin*, 89, 19-46.

Van der Tweel, L. H. & Verduyn Lunel, H. F. E. (1965). "Human visual responses to sinusoidally modulated light." *Electroencephalography and Clinical Neurology*, 18, 587-598.

Van Dusen, W. (1975). *The Presence of Other Worlds.* London: Wildwood House.

Walter, V. J. & Walter, W. G. (1949). "The central effects of rhythmic sensory stimulation." *Electroencephalography and Clinical Neurophysiology*, 1, 57-86.

Wickramasekera, I. E. (1988). *Clinical Behavioral Medicine: Some Concepts and Procedures.* New York: Plenum Press.

## Technical Notes

If you are using a compressed, or double-spaced hard drive, there are some special considerations. Recording high data rate waves (such as 44.1K/Stereo/16-bit) may severely slow down your system. If your system appears to lock up, press the Stop button, then try reducing the play/record buffer size in the Settings menu. Also be aware that the initial recording time left that is displayed when recording can be slightly off, since the amount of space audio data takes up on a compressed drive varies with the type of audio being recorded! Cool will try and estimate the time left every few seconds, so if you see a jump in time left (either a favorable jump, or a not so favorable decrease) it is because Cool is trying the best it can to estimate the remaining time.

For more accuracy in using any transformation, use 16-bit samples while working with waves. If your board only supports 8-bit samples, you can have them converted on the fly at playback time by choosing the "convert 16 to 8 bit" mode in Settings. To convert an 8-bit sample to 16-bit, open the waveform, copy the entire wave, then say "New" and choose 16-bit, and paste the waveform. Alternatively, you can open an 8-bit file as 16-bit by using File:Open As.

All temporary files begin with the tilde (~) symbol. The main Cool Edit temporary file containing the currently active waveform begins with ~COL. Undo information is saved in a file starting with ~NDO. Some other functions create files starting with ~CTE and ~CTM. All temporary files have the .TMP filename extension. The rest of the filename is chosen at random when the file is created. If there are no copies of Cool Edit running, none of these files should be present. If you find them, you can safely delete them, as long as Cool Edit is not running. These files can be left behind in extreme circumstances when Cool Edit crashes, or Windows unexpectadely quits while Cool Edit is running.

If you find you cannot load or save .WAV format waveforms, if you try to load a .WAV waveform file and the "Choose Sample Rate" dialog appears, or have any trouble with loading and saving in the proper format, then remove all copies of the .FLT files on your hard drive, and copy over the ones distributed with your current version of Cool Edit into the same directory as COOL.EXE. If different versions (older or newer) of the .FLT files are mixed with the distributed versions, these problems could arise. This problem could also result from running a separate program with the same name as the .FLT file (eg wave.flt will not be used if an application called wave.exe is running). If this is the case, simply rename the .flt file (eg rename wave.flt to wav.flt).

If the TEMP environment variable points to an invalid directory, some functions may fail. Be sure the TEMP environment variable is set to a valid directory with plenty (at least 1 meg) of hard drive space. Alternatively, you can add the TEMPOVERRIDE= line to the [Size] section of cool.ini (found in the Windows directory) and set it to your temporary drive and directory.

## About the Author

That's me, David Johnston. I have been working with various sound programs since about 1991, and have been toying with electronic noise makers (synthesizers) since the 70's. In high school I was even a pseudo-member of the band with a home-made programmable "sequencer" that I used to make Pac-Man noises for the song Pac Man Fever. I also started programming while in high school, when the school got a Commadore PET computer. Shortly thereafter I bought an Atari 800 (summer 1982) and started my programming career. I have since programmed on Apple ]['s, Commadore 64's, Macintoshes, IBM PC's (the original on up to today's), numerous mainframes and workstations. I attended Eastern Montana College for my first two years, then moved to the Seattle area and got my BS in Computer Science and Engineering from the University of Washington in 1991. I now work for a large software company, and write fun programs on my home PC when I get a chance. This is one of them! I use my ShareWare donations to help feed my technology habit, so expect to see more and more features added to Cool Edit.

I have tried to include all the wave editing features I have found useful, and am welcome to any suggestions for new waveform transformation functions, user interface, file formats (helps if you can provide a description of the new file format you want), and features. The "wave" function in this program is quite unique and I hope you check it out. It works only with stereo waveforms, and is the same function as in the freeware program out called MINDSYNC (wave.exe or wave87.exe) which I finished in July of 1992; See the Transform:Wave function for more details.

I hope you find lots of uses for Cool Edit, and find it fun and enoyable to use. Feel free to let me know what you think of it. It's because of input from numerous users out there that the program has evolved to what it is today. I have also written some other programs, but some of them may not be available on your local BBS. If you can't one you're interested in anywhere, I will gladly send you a copy for about $5 for the disk and postage.

**MindSync for DOS v.1.01** - This is the original brainwave synchronization program I wrote for DOS and the Sound Blaster. It only generates brainwave files in 22K stereo 8-bit VOC format. Brainwave synchronization is the same as the *Wave* function in Cool Edit.

**Kaleidoscope Screen Saver** - This is a great screen saver for use with Windows. Lots and lots of options let you customize, or completely randomize the kaleidoscopic variations. It's hard to describe the beauty of the living, flowing artwork that *is* the Kaleidoscope screen saver. Can also change patterns based on incoming music! Revive your old CD collection and actually *see* your music.

**After Dark Screen Savers** - I have written three after dark screen savers: Kaleidoscope (scaled down version of the Windows counterpart), Organic (fractal designs that look like cells), and String Arts (colorful designs resembling string art -- a popular artform of the 60's and 70's where colored string is strung between nails to make beautiful patterns).

**RPN Calculator** - Reverse Polish Notation calculator (like HP calculators). This calculator has dozens of great functions, including Hex/Binary/Decimal/Octal/n-ary base translations, trig functions, factorial, prime number, exponents and logs, and even biorhythms.

**Wallpaper Maker** - Create colorful wallpaper 'tiles' that look superb when tiled as Windows wallpaper. Never draws the same tile twice. See patterns in the tiles that are only visible when viewed as a group of tiles.

**Classic Collection** - Various Windows and DOS based graphic programs. If you're looking for some *classic* programs that are fun to use and watch, these may interest you. Use your mouse to control birth and death rates in a Game of Life, or place wagers on which species will win out in a different life simulation.  Steer your way through a warp speed starfield simulation.  Experiment with Iteraded Function Systems to grow plants and trees.  Weave your way through the Mandelbrot set (yes, I know, there are thousands of Mandelbrot generators out there now).  Shoot marbles around the screen, and experiment with the effects of gravity.  Generate screen mazes.  Draw spiro designs, watch string art designs, or point set fractal art.  Even includes a character based 'snow' program.

Cool Edit can support any number of file formats. New file formats can be written by following this API. A file filter is nothing more than a DLL with the following exported functions that Cool Edit calls to do the reading (decoding) and writing (coding) of audio data.

When Cool Edit starts, it checks the program directory for any files ending in .FLT. If it finds any, it checks to see if the file contains the function QueryCoolFilter, and if so, it calls that function. If the return value is C_VALIDLIBRARY, then it is assumed to be a valid file format DLL.

When reading an audio file, the function FilterUnderstandsFormat() is called first to see if your file filter can read the given file. If it can, then OpenFilterInput() is called, and then FilterCanReadSpecial() and FilterReadSpecial() if the QF_READSPECIALFIRST flag is set. Audio data is then read in chunks of the size specified by calling ReadFilterInput(). If QF_READSPECIALLAST is set, then the special read functions are called after all audio data is read in. Finally, the CloseFilterInput() function is called when all is complete. The functions FilterOptions(), FilterOptionsString(), and FilterGetFileSize() may be called at any time while the input file is open.

When writing an audio file, the function FilterGetOptions() may be called at any time if the QF_HASOPTIONSBOX flag is set so the user can choose any filter specific options (eg A-Law, mu-Law or Linear in the case of .AU files, or 2-bit, 3-bit, or 4-bit compression in the case of DVI compressed files). The GetSuggestedSampleType() function is called to ensure that the file filter can write data in the format that Cool Edit will be presenting. If a different sample type is specified, it will be converted to the desired type before being saved. OpenFilterOutput() is called to initiate the writing process. As in reading, the special write functions may be called before or after writing the audio data depending on the flags set. Audio data is written by calling the WriteFilterOutput() function in blocks of the size specified by your filter at open time. Once everything is written, CloseFilterOutput() is called and writing is complete.

The FilterGetOptions() function should call up a dialog box for the user to choose the format options. The dialog box template should be a resource in the file filter DLL, thus it can be totally defined by the author of the DLL.

The sample .AU file filter gives an example of writing a file filter that has multiple options. You can use these files as a template for creating your own file formats.

All functions should be declared as **FAR PASCAL __export**.

**Overview of Functions called by Cool Edit**

| | |
|---|---|
| QueryCoolFilter | Returns information about the type of file filter |
| FilterGetFileSize | Returns the number of audio data bytes that are allowed to read from the file. |
| FilterUnderstandsFormat | Returns TRUE if the given file is of a format that this filter understands |
| GetSuggestedSampleType | Returns the desired sample format for writing |
| FilterGetOptions | Brings up a dialog box to choose options for the filter if options supported |
| FilterOptions | Returns the options DWORD for the currently opened file |
| FilterOptionsString | Returns a readable text description of the filter options |
| FilterCanReadSpecial | Returns TRUE if this file format can read special extra non-audio data |
| FilterCanWriteSpecial | Returns TRUE if this file format can write special extra non-audio data |
| FilterReadSpecial | Reads the requested special non-audio data from the file |
| FilterWriteSpecial | Writes the special non-audio data to the file |
| OpenFilterInput | Open the specified file for reading |
| ReadFilterInput | Read a specified number of bytes from the input audio stream |
| CloseFilterInput | Close the input file |
| OpenFilterOutput | Open the specified file for writing |
| WriteFilterOutput | Write a specified number of bytes to the output audio stream |
| CloseFilterOutput | Close the output file |

## int QueryCoolFilter( lpcq ) — Required

**COOLQUERY far * lpcq** *Structure to be filled with all information pertaining to the file filter.*

This function should fill the COOLQUERY structure with information about the file filter.

**Returns:** C_VALIDLABRARY if successful, zero otherwise.

The COOLQUERY data structure is defined as:

| | |
|---|---|
| **char** szName[24] | Textual description of file filter that will show in the File Open and File Save dialog boxes. |
| **char** szCopyright[80] | Any copyright information you care to put in for your DLL. This information is displayed as the file is being loaded or saved. |
| **WORD** Quad32 | Use the R_xxxx constants ORed together to form the sample rates supported by this filter. |
| **WORD** Quad16 | 4-Channel, 16-bit samples |
| **WORD** Quad8 | 4-Channel, 8-bit samples |
| **WORD** Stereo8 | Stereo, 8-bit samples |
| **WORD** Stereo12 | Stereo, 12-bit samples |
| **WORD** Stereo16 | Stereo, 16-bit samples |
| **WORD** Stereo24 | Stereo, 24-bit samples |
| **WORD** Stereo32 | Stereo, 32-bit samples |
| **WORD** Mono8 | Mono, 8-bit samples |
| **WORD** Mono12 | Mono, 12-bit samples |
| **WORD** Mono16 | Mono, 16-bit samples |
| **WORD** Mono24 | Mono, 24-bit samples |
| **WORD** Mono32 | Mono, 32-bit samples |
| **DWORD** dwFlags | Use the QF_xxxx flags ORed together to provide special information about what functions are supported by your filter, etc. and so on. See below for a list of QF_xxxx constants. |
| **char** szExt[4]; | 3-character default extension, in caps, followed by a NULL character |
| **long** lChunkSize; | Size of chuncks (in bytes) prefered when Cool Edit calls your read and write functions |

Constants for Mono/Stereo/Quad 8/12/16/24/32 fields:

| | |
|---|---|
| **R_5500** | Low quality, 5500 Hz |
| **R_11025** | AM radio quality, 11 KHz |
| **R_22050** | Radio quality, 22 KHz |
| **R_32075** | FM radio quality, 32 KHz (also valid for 32000 Hz) |
| **R_44100** | CD quality, 44.1 KHz |
| **R_48000** | DAT quality, 48 KHz |

Constants for dwFlags field:

| | |
|---|---|
| **QF_RATEADJUSTABLE** | File filter can handle adjustable rates, not just the standards |
| **QF_CANSAVE** | File filter supports saving (writing) |
| **QF_CANLOAD** | File loading (reading) supported |
| **QF_UNDERSTANDSALL** | File filter can read any file at all (used for PCM) |
| **QF_READSPECIALFIRST** | Special information should be read before data |
| **QF_READSPECIALLAST** | Special information should be read after data |
| **QF_WRITESPECIALFIRST** | Special information should be written before data |
| **QF_WRITESPECIALLAST** | Special information should be written after data |
| **QF_HASOPTIONSBOX** | This format supports multiple options |
| **QF_NOASKFORCONVERT** | Set to bypass asking for conversion if original in different rate, ie auto convert sample type |
| **QF_NOHEADER** | Set if this is a raw data format with no header |

This function must return the value **C_VALIDLIBRARY** if everything is successful, otherwise it should return zero.

## BOOL FilterUnderstandsFormat( lpszFilename ) — Required

**LPSTR lpszFilename** File name of file to test for understanding of.

Cool Edit calls this function to determine if the given file can be read by the file filter. You may need to open the file and read in the first few bytes to verify that the header is correct for your format. For headerless formats, you may just need check the filename extension to determine file validity.

**Returns:** TRUE if this file filter understands the format of szFilename, FALSE otherwise.

## void GetSuggestedSampleType( lplSamprate, lplwBitsPerSample, lpwChannels )

Required if Writing

| | |
|---|---|
| **LONG FAR * lplSamprate** | Sample rate |
| **WORD FAR * lpwBitsPerSample** | Bits per Sample (8 or 16) |
| **WORD FAR * wChannels** | Channels (1 or 2) |

Called with the current format of the waveform that is about to be saved. If this format is not supported, return the closest format that *is* supported, and Cool Edit will prompt the user if they wish to convert to this format before saving. Return zero in any of the values to indicate that this parameter does not matter. For example, if this format handles all sample rates, set *lplSamprate to zero.

## HANDLE OpenFilterOutput( lpszFilename, lSamprate, wBitsPerSample, wChannels, lSize, lplChunkSize, dwOptions )

Required if Writing

| | |
|---|---|
| **LPSTR lpszFilename** | File to open for writing |
| **LONG lSamprate** | Sample rate in Hz |
| **WORD wBitsPerSample** | Sample size (currently 8 or 16) |
| **WORD wChannels** | Number of channels (currently 1 or 2) |
| **LONG lSize** | Total amount of data to be written, in bytes. *samples = lSize*wBitsPerSample/8/wChannels* |
| **LONG FAR * lplChunkSize** | Cool Edit will pass this value in *lBytes* to WriteFilterOutput() |
| **DWORD dwOptions** | Options string returned from FilterGetOptions(). Zero indicates the default should be used |

A user defined type should be allocated and the handle returned. The user defined type should contain all variables that are used in the writing of files of this format. It will be passed in too all functions pertaining to writing. The output file should be opened for writing, overwriting any existing file of the same name since the user has already confirmed that the file name is correct. The user defined type should at the very least contain a handle to the opened file. The **lplChunkSize* parameter should be filled with the desired size of a chunk of audio data that WriteFilterOutput() would like. Set **lplChunkSize* to 16384 if the chunk size doesn't matter. The dwOptions parameter contains any specific options that the user chose when FilterGetOptions() was called. If the user never called this funciton, dwOptions will be zero, in which case some default should be used. This function will only be called if the QF_CANLOAD flag is set.

**Returns:** Valid handle to some internal output file structure, NULL file not opened for output for any reason.

## void CloseFilterOutput( hOutput )

Required if Writing

| | |
|---|---|
| **HANDLE hOutput** | Handle of user defined output structure |

No more data is to be writtin out, and the file should be cleanly closed and the user defined structure referred to by hOutput should be freed. As far as Cool Edit is concerned, the file has completed. The user may have hit Cancel during the write process, in which case this function may be called prematurely. If this is the case, the file should be cleaned up as much as possible so it can still be read, otherwise it should be deleted.

## DWORD WriteFilterOutput( hOutput, lpbData, lBytes)

Required if Writing

| | |
|---|---|
| **HANDLE hOutput** | Handle of user defined output structure |
| **BYTE FAR *lpbData** | Data to be written (format was given in call to OpenFilterOutput() ) |
| **LONG lBytes** | Number of bytes in lpbData to be written |

All data in lpbData should be written out to the file. The file handle should be part of the user defined structure specified by hOutput.

**Returns:** Number of bytes actually written. Once this value is less than lBytes, or is zero, then Cool Edit will stop passing data to be written, and will call CloseFilterOutput().

## HANDLE OpenFilterInput( lpszFilename, lplSamprate, lpwBitsPerSample, lpwChannels, hWnd, lplChunkSize )

Required if Reading

| | |
|---|---|
| **LPSTR lpszFilename** | File to open for writing |
| **LONG FAR * lplSamprate** | Sample rate in Hz |
| **WORD FAR * lpwBitsPerSample** | Sample size (currently 8 or 16) |
| **WORD FAR * lpwChannels** | Number of channels (currently 1 or 2) |
| **HWND hWnd** | Parent window - use to display error messages if needed |
| **LONG FAR * lplChunkSize** | Cool Edit will pass this value in *lBytes* to ReadFilterInput() |

A user defined type should be allocated and the handle returned. The user defined type should contain all variables that are used in the reading of files of this format. It will be passed in too all functions pertaining to reading. The actual sample format should be filled into the lplSamprate, lpwBitsPerSample, and lpwChannels parameters. Fill lplChunkSize with the size of chunks you want ReadFilterInput to be called with. Use the hWnd value as a parent window if any custom dialogs or anything are used here. Custom dialogs for configuring should be called from the FIlterGetOptions() function instead of here. This function will only be called if the QF_CANSAVE flag is set.

**Returns:** A handle to the user defined structure used in reading, or NULL if the file could not be opened.

## DWORD FilterGetFileSize( hInput )

Required if Reading

| | |
|---|---|
| **HANDLE hInput** | Handle to user defined input structure, allocated during OpenFilterInput() |

The size of the audio data in the file, in bytes, should be returned. This function is only called after OpenFilterInput succeeds.

**Returns:** The size of the audio data in bytes.

## DWORD ReadFilterInput( hInput, lpbData, lBytes)

Required if Reading

| | |
|---|---|
| **HANDLE hOutput** | Handle of user defined input structure |
| **BYTE FAR *lpbData** | Filled with newly read in data |
| **LONG lBytes** | Number of bytes in lpbData that should be read |

lBytes bytes of audio data should be read into lpbData. The file handle should be part of the user defined structure specified by hInput.

**Returns:** Number of bytes actually read. Once this value is less than lBytes, or is zero, then Cool Edit will stop passing data buffers to be read into, and will call CloseFilterInput().

## void CloseFilterInput( hInput )

Required if Reading

| | |
|---|---|
| **HANDLE hInput** | Handle of user defined input structure |

No more data is going to be read in. The file should be cleanly closed and the user defined structure referred to by hInput should be freed. As far as Cool Edit is concerned, the file has completed reading in. The user may have hit Cancel during the read process, in which case this function may be called prematurely.

## DWORD FilterGetOptions( hWnd, hInst, lSamprate, wChannels, wBitsPerSample, dwDefaultOptions ) Optional

| | |
|---|---|
| **HWND hWnd** | Handle to use as parent window of options dialog box |
| **HINSTANCE hInst** | Instance of your DLL so you can access an options dialog box template |
| **LONG lSamprate** | Sample rate of audio that will be saved |
| **WORD wChannels** | Number of channels in audio that will be saved |
| **WORD wBitsPerSample** | Bits per sample in audio that will be saved |
| **DWORD dwDefaultOptions** | Previous options settings, or zero to indicate defaults should be set. |

If the QF_HASOPTIONSBOX flag is set, then the user may decide to change the file filter's parameters before saving. If so, this function is called. This function should in turn call up an options dialog box so the user may change the options. The options dialog box should default to the options specified by dwDefaultOptions. All options should fit within a single DWORD. The options DWORD should never be zero, since zero indicates that the default should be used.

**Returns:** Return zero if no options box is supported, otherwise return a valid DWORD containing all the options that the user chose. This options DWORD will be passed to OpenFilterOutput().

## DWORD FilterOptions( hInput ) Optional

| | |
|---|---|
| **HANDLE hInput** | Handle to user defined input structure |

If file filter supports multiple options, then this function should return the current options settings for the opened input file. This value will be used if the file is subsequently saved again in the same format, or if the Options dialog box is called up through FilterGetOptions().

**Returns:** Return zero if no options box is supported, othrwise return a valid DWORD containing the current options settings for the input file.

## DWORD FilterOptionsString( hInput, lpszString ) Optional

| | |
|---|---|
| **HANDLE hInput** | Handle to user defined input structure |
| **LPSTR lpszString** | String to hold text of options setting |

The lpszString parameter should be filled with a textual description of the current options settings. Currently this information is displayed for the user when "Show Info" is checked while choosing a file to open.

**Return:** Returns zero, and lpszString set to the empty string, if no options box is supported. Otherwise, a valid DWORD containing the current options settings should be returned.

## DWORD FilterCanWriteSpecial( hOutput, wType ) Optional

| | |
|---|---|
| **HANDLE hOutput** | Handle to user defined ouptut type |
| **WORD wType** | One of the SP_ types |

If the QF_WRITESPECIALFIRST or QF_WRITESPECIALLAST flag is set, this function is called to see if the requested type is handled by your file fitler. Some file formats support extra information besides just waveform data. If this is the case, the requests from Cool Edit to write the data can be asked before the audio data, or after, depending on the QF_ flag that is set.

**Return:** Zero if the requested type is not supported, or 1 if the type is supported by your file filter.

## DWORD FilterWriteSpecial( hOutput, wType, lpszData, dwSize ) Optional

| | |
|---|---|
| **HANDLE hOutput** | Handle to user defined ouptut type |
| **WORD wType** | One of the SP_ types |
| **LPSTR lpszData** | Data to be written |
| **DWORD dwSize** | Number of bytes of data to be written, includes null terminator for strings |

The special information presented by this function should be either written to the output file, or saved in the user defined output structure to be written out when CloseFilterOutput() is called. The data pointed to by lpszData is formatted differently depending on the type. Following is a list of the current types and formatting for lpszData:

| wType | Data formated as | Description |
|---|---|---|
| **SP_IART** | Zero terminated string | Original Artist string, single line |
| **SP_ICMT** | Zero terminated string w/ LF's | Comments string, multiple lines |
| **SP_ICOP** | Zero terminated string | Copyright string, single line |
| **SP_ICRD** | Zero terminated string | Creation date string, single line |
| **SP_IENG** | Zero terminated string | Engineers string, single line |
| **SP_IGNR** | Zero terminated string | Genre string, single line |
| **SP_IKEY** | Zero terminated string | Key words string, single line |
| **SP_IMED** | Zero terminated string | Original Medium string, single line |
| **SP_INAM** | Zero terminated string | Name string, single line |
| **SP_ISFT** | Zero terminated string | Software Package string, single line |
| **SP_ISRC** | Zero terminated string | Digitization source string, single line |
| **SP_ITCH** | Zero terminated string | Digitizer/Technition string, single line |
| **SP_ISBJ** | Zero terminated string w/ LF's | Subject string, multiple lines |
| **SP_ISRF** | Zero terminated string | Source Reference string, single line |
| **SP_DISP** | <clipboard type> <data> | Clipboard formatted data (eg CF_BITMAP, CF_TEXT) |
| **SP_CUE** | Array of struct cue_type's | Cue list |
| **SP_LTXT** | Array of struct ltxt_type's | Additional information for cue list entries, like length information |
| **SP_NOTE** | List of <id><zero-term-string> | Comments for cue list entries |
| **SP_LABL** | List of <id><zero-term-string> | Labels for cue list entries |
| **SP_PLST** | Array of struct play_type's | Play List |

**Returns:** The number of bytes processed

## DWORD FilterCanReadSpecial( hInput, wType ) Optional

| | |
|---|---|
| **HANDLE hOutput** | Handle to user defined ouptut type |
| **WORD wType** | One of the SP_ types |

If the QF_READSPECIALFIRST or QF_READSPECIALLAST flag is set, this function is called to see if the requested type is handled by your file fitler. Some file formats support extra information besides just waveform data. If this is the case, the requests from Cool Edit to read the data can be asked before or after the audio data, depending on the QF_ flag that is set.

**Return:** Zero if the requested type is not supported, or the number of bytes needed for the specific type.
For the SP_CUE type, return the number of cue list entries.
For the SP_PLST type, return the number of play list entries.

## DWORD FilterReadSpecial( hInput, wType, lpszData, dwSize ) Optional

| | |
|---|---|
| **HANDLE hOutput** | Handle to user defined ouptut type |
| **WORD wType** | One of the SP_ types |
| **LPSTR lpszData** | Data buffer |
| **DWORD dwSize** | Number of bytes of data to read into buffer |

If any special information exists in the file and is handled by your file format, you can either read it all in during OpenFilterInput(), or when it is requested through this function. Cool Edit will call this function either before or after the audio data is read depending on which of the QF_READSPECIALFIRST or QF_READSPECIALLAST flags are set. The number of bytes requested should be read in from the file, or copied from the user defined input structure. The various types and the format of lpszData for each type are listed in the FilterWriteSpecial() function.

**Returns:** The number of bytes actually read.

## Structures

Following are the structures that are used in reading and writing files. Most have to do with the transfer of special types of extra data (data that is not audio). For more information on specific special types, see the RIFF specification, since the format of the data closely parallels that in the specification. Cool Edit actually only supports 8 and 16 bit audio in mono and stereo formats - any sample rate. The COOLQUERY type allows for other formats though.

```
typedef DWORD FOURCC;              // a four character code placed into a single DWORD (lsb first)

struct cue_type
{       DWORD dwName;              // A unique value identifying this cue entry
        DWORD dwPosition;          // Not used by Cool Edit
        FOURCC fccChunk;           // currently always 'data'
        DWORD dwChunkStart;        // always zero
        DWORD dwBlockStart;        // always zero
        DWORD dwSampleOffset;      // Sample offset of cue position
};

struct ltxt_type
{       DWORD dwName;              // A cue dwName to identify which cue entry this info belongs to
        DWORD dwSampleLength;      // Number of samples long for the cue entry (zero for a cue point, more for a range)
        FOURCC fccPurpose;         // (fourcc code, like 'mark')
        WORD wCountry;             // Zero to ignore
        WORD wLanguage;            // Zero to ignore
        WORD wDialect;             // Zero to ignore
        WORD wCodePage;            // Zero to ignore
};

struct play_type
{       DWORD dwName;              // A cue dwName to identify which cue entry to play
        DWORD dwLength;            // Length in samples (range starts at cue's start point to cue start point + dwLength)
        DWORD dwLoops;             // Number of times to loop
};

typedef struct coolquery_tag
{       char szName[24];           // Text description of file filter that shows in File Open and Save dialogs
        char szCopyright[80];      // Any copyright info, will display while file loads or saves
        WORD Quad32;               // Bit field of supported rates of quadraphonic 32-bit audio
        WORD Quad16;               // Bit field of supported rates of quadraphonic 16-bit audio
        WORD Quad8;                // Bit field of supported rates of quadraphonic 8-bit audio
        WORD Stereo8;              // Bit field of supported rates of stereo 8-bit audio
        WORD Stereo12;             // Bit field of supported rates of stereo 12-bit audio
        WORD Stereo16;             // Bit field of supported rates of stereo 16-bit audio
        WORD Stereo24;             // Bit field of supported rates of stereo 24-bit audio
        WORD Stereo32;             // Bit field of supported rates of stereo 32-bit audio
        WORD Mono8;                // Bit field of supported rates of mono 8-bit audio
        WORD Mono12;               // Bit field of supported rates of mono 12-bit audio
        WORD Mono16;               // Bit field of supported rates of mono 16-bit audio
        WORD Mono24;               // Bit field of supported rates of mono 24-bit audio
        WORD Mono32;               // Bit field of supported rates of mono 32-bit audio
        DWORD dwFlags;             // Various flags (see QueryCoolFIlter())
        char szExt[4];             // 3-letter filename extension for file format, zero terminated
        long lChunkSize;           // Desired data block size for reading or writing (in bytes)
} COOLQUERY;
```

## Definitions

```
#define R_5500      1                    // Rate flags for supported sample types bitfields in COOLQUERY
#define R_11025     2
#define R_22050     4
#define R_32075     8
#define R_44100     16
#define R_48000     32
#define R_88200     64

#define C_VALIDLIBRARY 134               // Returned from QueryCoolFilter() to designate a valid file format

                                               // Flags for dwFlags field of COOLQUERY
#define QF_RATEADJUSTABLE         1       // All sample rates are valid, not just the standard "R_" ones.
#define QF_CANSAVE                2       // Set if file filter can write files
#define QF_CANLOAD                4       // Set if file filter can read files
#define QF_UNDERSTANDSALL         8       // Set if file filter reads any file format whatsoever (reserved for RAW PCM)
#define QF_READSPECIALFIRST       16      // Set if extra info should be requested for before reading audio
#define QF_READSPECIALLAST        32      // Set if extra info should be requested for after reading audio
#define QF_WRITESPECIALFIRST      64      // Set if extra info should be written for before writing audio
#define QF_WRITESPECIALLAST       128     // Set if extra info should be written for after writing audio
#define QF_HASOPTIONSBOX          256     // Set if file format supports multiple options
#define QF_NOASKFORCONVERT        512     // Set to convert sample type automatically, see GetSuggestedSampleType()
#define QF_NOHEADER               1024    // Set if this file format does not have a header, if it is just raw data

#define SP_IART     20
#define SP_ICMT     21
#define SP_ICOP     22
#define SP_ICRD     23
#define SP_IENG     24
#define SP_IGNR     25
#define SP_IKEY     26
#define SP_IMED     27
#define SP_INAM     28
#define SP_ISFT     29
#define SP_ISRC     30
#define SP_ITCH     31
#define SP_ISBJ     32
#define SP_ISRF     33
#define SP_DISP     34          // Clipboard formatted data
#define SP_CUE      40          // returns number of cues of size cue_type
#define SP_LTXT     41          // returns number of adtl texts of size 8 (4,id and 4,len)
#define SP_NOTE     42          // returns LO=size, HI=number of strings (sz sz sz...)
#define SP_LABL     43          // returns LO=size, HI=number of strings (sz sz sz...)
#define SP_PLST     44          // returns number of playlist entries size play_type
```

Cool Edit can support any number of transform and generate effects. New effects can be written by following this API. An effects module is nothing more than a DLL with the following exported functions that Cool Edit calls to do the reading and writing of audio data.

When Cool Edit starts, it checks the program directory for any files ending in .XFM. If it finds any, it checks to see if the file contains the function QueryXFM, and if so, it calls that function. If the return value is XFM_VALIDLIBRARY, then it is assumed to be a valid effects module DLL.

The only functions that are required are QueryXfm(), XfmInit(), XfmDestroy(), XfmSetup(), XfmDo(), and some sort of DIALOGMsgProc() for the settings dialog. A custom structure should be defined to hold all effect specific data. This structure must also be communicated to Cool Edit so that various variables in the structure can be filled in automatically if you are using the Presets functions. The COOLINFO structure contains many functions that can be called at any time. These functions can be accessed easily by using the macros defined in xfmsdll.c. Functions include reading and writing audio data, progress meter control, cutting and inserting blank data, preset handling, graph handling, FFT functions, and functions for calling other XFM modules.

An internal data structure is used, and defined by the author of the DLL, for storing any relavent information pertaining to the effect. The structure's format is communicated to Cool Edit through the szStructDef element of XFMQUERY during QueryXfm(). Never use any global variables - any variables you need should either be defined as part of the effect's internal data structure that you define, or allocated locally to the procedure that is using them. This is important since there may be more than one process using the DLL at any given time.

If you follow the sample code, and use it as a starting point for your own effects module, then you can follow some of these guidelines. Use the DialogToStruct() function to copy data from your instance data structure to the dialog box. Use the StructToDialog() function to copy data from the dialog box back to your data structure. You can leave the CopyToDlgItem() and CopyFromDlgItem() functions essentially unchanged (except for inserting the name of your custom structure). These two functions copy the handles to your custom data structure, and to the COOLINFO structure to a dialog control so that the instance information can be kept for your instance. This way, if mutliple copies of Cool Edit are using the same effects DLL, then the data will not get scrambled between them. The CopyToDlgItem() function is called during the dialog box's WM_INITDIALOG, and the CopyFromDlgItem() function is called before any of the data in the custom structure or COOLINFO structure is needed. Include the file xfmsdll.c to gain access to all the COOLINFO structure functions that are provided to you by Cool Edit.

All XFM functions should be declared as **FAR PASCAL __export**.

## int QueryXfm( lpxq ) Required

| | |
|---|---|
| **XFMQUERY FAR * lpxq** | Structure to be filled with all information pertaining to the effect module |

This function should fill the XFMQUERY structure with information about the effect module.

**Returns:** XFM_VALIDLABRARY if successful, zero otherwise.

The XFMQUERY data structure is defined as:

| | |
|---|---|
| **char** szName[40] | Text name of the effect that will appear in the menu bar |
| **char** szCopyright[80] | Any copyright information you care to put in for your DLL |
| **char** szToolHelp[70] | Text to be shown in toolbar quick-help |
| **WORD** wSupports | Combination of XFM_ flags for mono and stereo 8 or 16 bit |
| **DWORD** dwFlags | Combination of the XF_ flags for describing module |
| **DWORD** dwUserDataLength | Length of transform's internal data structure |
| **char** szStructDef[48] | Array of chars representing types of data in internal data structure |
| **char** szPresetDef[48] | Array of 'y' or 'n' representing whether or not to include data item in presets saving |

Constants for the wSupports field:

| | |
|---|---|
| **XFM_MONO8** | Effect supports Mono 8-bit data |
| **XFM_STEREO8** | Effect supports Stereo 8-bit data |
| **XFM_MONO16** | Effect supports Mono 16-bit data |
| **XFM_STEREO16** | Effect supports Stereo 16-bit data |

Constants for the dwFlags field:

| | |
|---|---|
| **XF_TRANSFORM** | This function works with highlighted audio, and belongs in the Transform menu |
| **XF_GENERATE** | This function generates new wave data, and belongs in the Generate menu |
| **XF_USESPRESETSAPI** | Has a presets box (and uses the presets/scripts functions) |
| **XF_USESGRAPHAPI** | Has one or more graph controls (and uses the built in graph control functions) |
| **XF_MUSTHIGHLIGHT** | If set, function only enabled if a selection is highlighted (most common) |
| **XF_MUSTHAVECLIP** | If set, function is grayed if nothing on clipboard |
| **XF_MODIFIESTOENDOFVIEW** | Set this if function modifies data outside user's given selection to end of view |
| **XF_MODIFIESTOENDOFFILE** | Set this if function modifies data outside user's given selection to end of file |
| **XF_USESFFTAPI** | Uses Cool's FFT functions |
| **XF_NOSINGLEEDIT** | If set, function is grayed if editing one channel of a stereo waveform (do this if effect changes the size of the highlighted data, eg. stretching) |

Characters used in the structure definition (szStructDef) to tell Cool Edit the data types in the internal effect data structure:

| | |
|---|---|
| **'c'** | **char** |
| **'i'** | 16-bit int or **WORD** |
| **'l'** | 32-bit long or **DWORD** |
| **'f'** | 4-byte **float** |
| **'d'** | 8-byte **double** floating point |
| **'g'** | **HANDLE** to a graph |
| **'h'** | **HANDLE** to globally allocated memory |
| **'s'** | Array of **256 chars** |

## BOOL XfmInit( ci ) Required

| | |
|---|---|
| **COOLINFO FAR * ci** | Cool Edit info structure pointer |

This function should do all initialization of the internal data structure that contains all data pertaining to the effect. The *hUserData* member of *ci* is a handle to this data. Basically, this function should lock the hUserData member, casting it to the internal data type, and fill all the members with default info, and finally unlock the handle.

**Returns:** TRUE if all was initialized OK, FALSE if there was an error.

## BOOL XfmDestroy( ci ) Required

| | |
|---|---|
| **COOLINFO FAR * ci** | Cool Edit info structure pointer |

All internally allocated data should be freed (eg globally allocated handles) or destroyed (eg graphs). Use the *hUserData* member of *ci* to access any internal data.

**Returns:** TRUE if all was sucessfully destroyed, FALSE otherwise.

## BOOL XfmSetup( hWnd, hInst, ci ) Required

| | |
|---|---|
| **HWND hWnd** | Handle to a parent window for your setup dialog box |
| **HINSTANCE hInst** | Instance handle for this DLL so you can access resources |
| **COOLINFO FAR * ci** | Cool Edit info structure pointer |

This function should call up a settings dialog. The dialog box template should be compiled with the DLL, and can be accessed using the *hInst* parameter. Below is a sample XfmSetup() function that should suffice for any module. IDD_TRANSFORM is the dialog box ID, and the dialog box message proc is exported with ordinal 100. The lParam given to the dialog box proc during WM_INITDIALOG will be a far pointer to the COOLINFO structure.

```
BOOL FAR PASCAL __export XfmSetup(HWND hWnd, HINSTANCE hInst, COOLINFO far *ci)
{       int nRc;
        FARPROC lpfnDIALOGMsgProc;
        lpfnDIALOGMsgProc = GetProcAddress(hInst,(LPCSTR)MAKELONG(100,0));
        nRc = DialogBoxParam(   (HINSTANCE)hInst,(LPCSTR)MAKEINTRESOURCE(IDD_TRANSFORM),
                                (HWND)hWnd, (DLGPROC)lpfnDIALOGMsgProc,(DWORD)ci);
        return nRc;
}
```

This setup function is called when the user chooses this effect from the menubar. The dialog box routine should call EndDialog with TRUE if the user presses OK, or FALSE if they hit Cancel. Special care must be taken if more than one settings dialog is to be open at one time (if two instances of Cool Edit each have your effect dialog open). See the example's CopyToDlgItem() and CopyFromDlgItem() functions which copy the relavent instance data to a hidden dialog box text control.

**Returns:** TRUE to continue on and call XfmDo to run the function, or FALSE if the user hit Cancel in the dialog box.

## BOOL XfmDo( ci ) Required

| | |
|---|---|
| **COOLINFO FAR * ci** | Cool Edit info structure pointer |

This function is what does all the actual work. The COOLINFO structure contains all information necessary for doing the effect, including the portion of the wave that was highlighted, and a handle to this effect's specific internal data structure that should have been filled in during the Setup function.

**Returns:** TRUE if all went OK, or FALSE if there was an error.

## void DialogToStruct( ci, hWndDlg, lpVars ) Recommended
## void StructToDialog( ci, lpVars, hWndDlg )

| | |
|---|---|
| **COOLINFO FAR * ci** | Cool Edit info structure pointer |
| **HWND hWndDlg** | Settings dialog window handle |
| **EFFECTVARS FAR * lpVars** | Pointer to internally defined structure |

These functions should transfer data from the dialog box to the user defined data structure, and back. For example, an effect that has a Volume setting would get the text from the "ID_VOLUME" edit control in the dialog and copy it to the lpVars->wVolume parameter when calling DialogToStruct. It would set the "ID_VOLUME" control text to a string containing a value derived from lpVars->wVolume when calling StructToDialog. Since these operations are done frequently, it is best to have them as a separate function. StructToDialog() will be called during the WM_INITDIALOG call, and DialogToStruct() will be called during processing of IDOK.

## void CopyToDlgItem( hWndDlg, iControl, ci, hVars, lpVars ) Recommended
## void CopyFromDlgItem( hWndDlg, iControl, lpci, phVars, plpVars )

| | |
|---|---|
| **HWND hWndDlg** | Settings dialog window handle |
| **int iControl** | ID of control to store info into (control should be text type and hidden) |
| **COOLINFO FAR * ci** | Cool Edit info structure pointer |
| **COOLINFO FAR ** lpci** | Pointer to Cool Edit Info Structure pointer |
| **HANDLE hVars** | Handle to internally defined structure |
| **HANDLE * phVars** | Pointer to handle to internally defined structure |
| **EFFECTVARS FAR * lpVars** | Pointer to internally defined structure |
| **EFFECTVARS FAR ** plpVars** | Pointer to pointer to internally defined structure |

These functions basically copy the handles and pointers used most frequently to a dialog control, and copy data back from the control to the appropriate handles and pointers. During the WM_INITDIALOG, the internal data is locked and the pointer to the data is saved in a control's text field. Whenever any other messages are processed that need this information, it is copied back from the control to the locally declared pointers and handles. Following is the most common implementation for these two functions. The control ID passed into these functions should represent a hidden text field in the dialog box.

```
void CopyToDlgItem(HWND hWndDlg, int iControl,COOLINFO far *ci,HANDLE hMyVars,  COMPRESS FAR *lpMyVars)
{       char m[80];
        wsprintf(m,"%ld,%d,%ld",lpAmp,hCompress,ci);
        SetDlgItemText(hWndDlg,iControl,m);
}

void CopyFromDlgItem(HWND hWndDlg, int iControl,  COOLINFO far **ci, HANDLE *hMyVars, COMPRESS FAR **lpMyVars)
{       char m[80];
        char *cursor;
        GetDlgItemText(hWndDlg,iControl,m,80);
        if (!m[0])
        {       *lpMyVars=NULL;
                *hMyVars=0;
                *ci=NULL;
                return;
        }
        cursor=m;
        *lpMyVars=(EFFECTVARS FAR *)longfromstring(&cursor);
        *hMyVars=(HANDLE)longfromstring(&cursor);
        *ci=(COOLINFO far *)longfromstring(&cursor);
}
```

## COOLINFO Structure

The COOLINFO structure contains everything you may need for performing a transform effect on existing data, or to create new data. Following are explanations of the various fields of COOLINFO, though they are not in the same order as declared in the xfms.h file. The procedure pointers have not been mentioned - see the following section for info on all procedures.

| | |
|---|---|
| **WORD wChannels** | Number of channels in audio data (1 or 2) |
| **WORD wBitsPerSample** | Bits per sample (8 or 16) |
| **WORD wBlockAlign** | Bytes per sample (1, 2, or 4) |
| **long lSamprate** | Sample rate of given audio data |
| **HANDLE hUserData** | Handle to user's internal data structure |
| **DWORD dwLoSample** | Starting sample offset to transform |
| **DWORD dwHiSample** | Ending sample offset to transform |
| **DWORD dwRightSample** | Rightmost visible sample on the current display |
| **DWORD dwLeftSample** | Leftmost visible sample on the current display |
| **FARPROC lpTestFunction** | Nothing. |
| **XFMQUERY FAR * cq** | Structure returned by QueryXfm() |
| **char far * szIniFile** | INI file to use for storing any data you might want to store. |
| **HFONT hFont** | Handle to a small font that you may use in dialog boxes |
| **BOOL bReplacesHighlightedSelection** | If set, when generating audio, newly generated audio will replace highlighted sel. |
| **DWORD dwInsertBlankSamples** | If nonzero, this many blank samples will be inserted after OK is hit. |
| **BOOL bHasCoprocessor** | Set if user has a coprocessor |
| **HINSTANCE hInst** | Instance of this DLL |
| **int iScriptFile** | -1 if no script running, else script is running |
| **int iScriptDialogStop** | Nonzero if "Pause at Dialogs" set for Scripts |
| **BOOL FAR * lpProgressCanceled** | Pointer to progress canceled flag. If flag is set, user hit Cancel in progress meter. |

## int ReadData( ci, hpData, lOffset, lSize ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR * ci** | Cool Edit info structure pointer |
| **char huge * hpData** | Buffer to read into |
| **long lOffset** | Offset into user's wave data |
| **long lSize** | Number of bytes to read |

Read **lSize** bytes into the buffer **hpData** from the user's waveform data starting at **lOffset** bytes into user's data. The entire amount is always read. Data at offsets below zero, or greater than the length of the file are filled with zeroes for 16-bit data, or 128's for 8-bit data. Remember that the this functions works in units of *bytes*, not samples, and that the formula for converting samples to bytes is *Bytes = Samples * wBitsPerSample * wChannels / 8*.

Returns: 0 if all went OK, otherwise an error occurred.

## int WriteData( ci, hpData, lOffset, lSize ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR * ci** | Cool Edit info structure pointer |
| **char huge * hpData** | Buffer to write out |
| **long lOffset** | Offset into user's wave data |
| **long lSize** | Number of bytes to write |

Write **lSize** bytes from the buffer **hpData** to user's waveform data starting at **lOffset** bytes into user's data. The entire amount is always written. Data at offsets below zero are ignored, and data at offsets greater than the length of the file extend the length of the user's file. Remember that the this functions works in units of *bytes*, not samples, and that the formula for converting samples to bytes is *Bytes = Samples * wBitsPerSample * wChannels / 8*.

Returns: 0 if all went OK, otherwise an error occurred.

## void ProgressCreate( ci, cszText, hwndParent ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| LPCSTR cszText | Progress message to display above meter |
| HWND hwndParent | Parent window of meter display |

Create a progress meter to show the user how much of the function has been processed. This meter automatically estimates the total time the function is anticipated to take. **cszText** contains text that will be displayed at the top of the meter window, for example, "Flanging Selection...". Set **hwndParent** to NULL to have the Cool Edit main window be the parent (most common).

## BOOL ProgressMeter( ci, dwCurrent, dwTotal ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **DWORD dwCurrent** | Measure of current amount done |
| **DWORD dwTotal** | Measure of total amount to do |

The progress bar is updated with the percentage *100 * dwCurrent / dwTotal*. Usually **dwTotal** is the total number of bytes that your function is going to process, and **dwCurrent** is the number of bytes processed so far when the function is called. The progress meter must have been created with **ProgressCreate()** before calling this function.

Returns: FALSE if user hit Cancel, TRUE if all is going OK.

## void ProgressDestroy( ci ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |

Close the progress meter box. Always remember to close the progress meter if you created one when processing is finished.

## void CenterDialog( ci, hWndDlg ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HWND hWndDlg** | Handle of settings dialog |

Call this function in the WM_INITDIALOG section of your dialog's settings window to center the dialog with respect to the Cool Edit window that called it.

## BOOL PresetsInit( ci, hWndDlg, cszGroupName ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HWND hWndDlg** | Handle of settings dialog |
| **LPCSTR cszGroupName** | Name of function |

This should be called in the WM_INITDIALOG section of your dialog's settings window if presets are used. **cszGroupName** identifies a unique name for your function which is used as a key in the INI file for keeping the preset information. This is usually the same as the name of the function given in the XFMQUERY structure. The **szStructDef** array of XFMQUERY contains an array of characters representing each of the data items in the user defined structure. **szPresetDef** contains an array of 'y' and 'n' characteres which specify whether or not to include the associated user data item in the preset. For example, a user defined structure with 4 integers (szStructDef="iiii") of which only the first 3 should be saved when added to the presets would have szPresetDef equal to "yyyn".

The dialog box must have the following IDs for various preset controls:

```
#define ID_PRESETS       1005    // Single selection sorted list box with Notify flag set
#define ID_PRESETNAME    1006    // Edit control for user to enter new preset name to add or delete
#define ID_ADD           325     // Button with text "Add"
#define ID_DEL           1109    // Button with text "Del"
```

Returns: 0 if no presets yet exist, or the number of presets if some do exist. (you can cast the BOOL return type to int).

## BOOL HandleID_PRESETS ( ci, hWndDlg, cszGroupName, lParam ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HWND hWndDlg** | Handle of settings dialog |
| **LPCSTR cszGroupName** | Name of function |
| **LPARAM lParam** | lParam passed to settings dialog |

Call this function in response to the WM_COMMAND message when wParam is ID_PRESET. The function returns TRUE if the user chose a new preset item (double clicked on a name in the presets box). The **StructToDialog()** procedure should be called if this is the case to update all the dialog controls with the new data.

```
case ID_PRESETS:
    if ( HandleID_PRESETS( ci, hWndDlg, TRANSFORMNAME, lParam ) )
    {       StructToDialog( ci, lpMyVars, hWndDlg );
    }
    break;
```

Returns: TRUE if user chose a new preset item, FALSE otherwise.

## void HandleID_ADD ( ci, hWndDlg, cszGroupName ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HWND hWndDlg** | Handle of settings dialog |
| **LPCSTR cszGroupName** | Name of function |

Call this function in response to the WM_COMMAND message when wParam is ID_ADD. The **DialogToStruct()** function should be called before calling **HandleID_ADD()** to fill the COOLINFO structure with the information in the dialog box controls.

```
case ID_ADD:
    DialogToStruct(ci, hWndDlg, lpMyVars);
    HandleID_ADD(ci, hWndDlg, TRANSFORMNAME);
    break;
```

## void HandleID_DEL ( ci, hWndDlg, cszGroupName ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HWND hWndDlg** | Handle of settings dialog |
| **LPCSTR cszGroupName** | Name of function |

Call this function in response to the WM_COMMAND message when wParam is ID_DEL.

```
case ID_DEL:
      HandleID_DEL(ci, hWndDlg, TRANSFORMNAME);
      break;
```

## void HandleID_PRESETNAME ( ci, hWndDlg ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HWND hWndDlg** | Handle of settings dialog |

Call this function in response to the WM_COMMAND message when wParam is ID_PRESETNAME.

```
case ID_PRESETNAME:
      HandleID_PRESETNAME(ci, hWndDlg);
      break;
```

## HANDLE GraphCreate ( ci, iLeft, iRight, iMin, iMax, iLeftVal, iRightVal )COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **int iLeft** | Leftmost graph coordinate (along x axis) |
| **int iRight** | Rightmost graph coordinate |
| **int iMin** | Minimum graph value (along y axis) |
| **int iMax** | Maximum graph value |
| **int iLeftVal** | Value of leftmost point (must be between iMin and iMax) |
| **int iRightVal** | Value of rightmost point (must be between iMin and iMax) |

This function returns an handle to a graph that is passed to subsequent graph functions. Graphs are usually created in the **XfmInit()** function, and destroyed in the **XfmDestroy()** function. A graph control allows the user to specify any number of points connected by a single line that has only one *y* value for every *x* value.

Returns: Handle to graph if successful, or NULL if no graph was created.

## void GraphSetDialog ( ci, hGraph, hWndDlg, uiControl, uiDisplay ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |
| **HWND hWndDlg** | Handle of settings dialog |
| **UINT uiControl** | ID of rectangle control defining boundaries of graph |
| **UINT uiDisplay** | ID of static text control for displaying graph point data |

This should be called in the WM_INITDIALOG section of your dialog's settings window to attach dialog controls to the graph. The graph must know of a rectangle it can be drawn inside of, and of a text control for displaying graph relevant information, and of the handle to the dialog box itself. uiControl is the ID of a Frame type picture control in the dialog box, while uiDisplay is the ID of a left aligned text control in the dialog box. The graph will automatically fit itself inside the bounds of the rectangle.

## int GraphCount ( ci, hGraph ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |

The user can create new points in a graph at any time. This function returns the total number of points in the graph.

Returns: The number of points in the graph.

## POINT GraphGetPoint ( ci, hGraph, iIndex ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |
| **int iIndex** | Point to get (0 to **GraphCount()** - 1) |

Returns a point structure (whose members are .x and .y) that contains the x and y coordinates of the point at the given index. Index values range from 0 for the left hand point on up to the number of points in the graph minus one for the rightmost point.

Returns: x and y value of point in graph.

## void GraphSetPoint ( ci, hGraph, int iIndex, ptPoint, bEndPoint ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |
| **int iIndex** | Index of point to set |
| **POINT ptPoint** | Coordinates of point to add or set |
| **BOOL bEndPoint** | TRUE if this point is the rightmost point (as in adding new points to the end) |

Call this function to add new points or change the position of points in the graph. Not used very often. You can build a set of points for a graph by using this function, but usually the user is the one who creates the points by clicking and dragging. If the point you are adding is a new rightmost point, indicate by setting bEndPoint to TRUE.

## double GetValueAt ( ci, hGraph, fXValue ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |
| **double fXValue** | X coordinate on graph |

Returns the y value at any given point along the x axis of the graph. Values between points on the graph are linearly interpolated, so no matter what x value is given (provided it is between the graph's left and right values given at creation time), a valid value is returned. The function works with double precision floating point values since the y value is interpolated, and may not be an integral value.

Returns: The y value at the given x coordinate.

## void GraphDraw ( ci, hGraph, hDC ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |
| **HDC hDC** | Device Context of dialog box for drawing graph into |

This function should be called in the WM_PAINT routine of the settings window. The hDC should be the same hDC that was returned from BeginPaint(). The code snippet blow illustrates getting the pointer to the user defined structure (lpMyVars), and then getting the dialogs device context, drawing the graph, end ending the paint routine. The graph is a HANDLE type in the users structure, and has the code 'g' associated with it in the definition variable **szStructDef**.

```
case WM_PAINT:
{     HDC hDC;
      PAINTSTRUCT ps;
      CopyFromDlgItem( hWndDlg,IDC_STORAGE, &ci, &hMyVars, &lpMyVars );
       hDC=BeginPaint( hWndDlg, &ps );
       GraphDraw( ci, lpMyVars->hGraph, hDC );
       EndPaint( hWndDlg, &ps );
       return FALSE;
}
```

## void GraphClear ( ci, hGraph )
COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |

Clear all the points in the graph except for the leftmost and rightmost endpoints.

## void GraphInvert ( ci, hGraph )
COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |

Invert the graph if the graph is indeed invertable. The graph is invertable if there is exactly one x point for every y point. When the graph is inverted, every x value is swapped with every y value. The property that there is only one y value for each x value must still hold. Sorry, but no indication is returned if the function inverted the graph or not.

## void GraphCopy ( ci, hGraphDest, hGraphSource )
COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraphDest** | Handle to graph to copy into |
| HANDLE hGraphSource | Handle of graph to copy from |

Copy the contents of the source graph to the destination graph. Both graphs must have already been created using **GraphCreate()**.

## void GraphSetDblClkScales ( ci, hGraph, fScx, fOfx, fMagx, fScy, fOfy, fMagy, wStyle )
COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |
| **double fScx** | Scaling factor for X coordinate |
| **double fOfx** | Offset for X coordinate |
| **double fMagx** | Precision of X coordinate |
| **double fScy** | Scaling factor for Y coordinate |
| **double fOfy** | Offset for Y coordinate |
| **double fMagy** | Precision of Y coordinate |
| **WORD wStyle** | Style (set to 0) |

Set the scaling factors that the actual points will be modified by before being displayed in the text box. The text box whose ID was given in the **GraphSetDialog()** call will display the value of the point when a point is selected. The value displayed is not the same as the integer value of the point, but a value base on the following formula: *DisplayValue = PointValue * fScx + fOfx*. The value of the point is multiplied by the scaling factor and added to the offset to get the final value displayed to the user. The magnitude/precision value tells how many decimal places to round the value off to. For example, fMagx of 10 will round values to one decimal place (eg 3.1), while fMagx of 100 will round values to two decimal places (eg 3.14). Values that are not powers of 10 will round according, for example fMagx of 4 will give values such as "4", "4.25", "4.5", and "4.75".

The actual data displayed to the user will be this value along with the units given in the **GraphSetDblClkNames()** function. The wStyle parameter should always be zero. If it is set to 1, then the final value is converted to decibels (eg 0.5 becomes -3, and 2.0 becomes +3).

## void GraphSetDblClkNames ( ci, hGraph, cszXText, cszXUnits, cszYText, cszYUnits ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |
| **LPCSTR cszXText** | Name for values along x axis (going left and right) |
| **LPCSTR cszXUnits** | Units for values along x axis |
| **LPCSTR cszYText** | Name for values along y axis (going up and down) |
| **LPCSTR cszYUnits** | Unints for values along y axis |

Set the names for the values along the x and y axes, for example, "Input Signal Level" or "Frequency", and the names for the units of these, for example, "dB" or "Hz".

## int GraphHandleWM_LBUTTONDOWN ( ci, hGraph, ptCursor ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |
| **POINT ptCursor** | Coordinate of mouse down point (gained from lParam) |

Call this function in response to the WM_LBUTTONDOWN message in the settings dialog. The lParam parameter can be converted to a point since the LOWORD of lParam is the x coordinate, and the HIWORD of lParam is the y coordinate.

```
case WM_LBUTTONDOWN:
{       POINT here;
        int whichpoint;
        CopyFromDlgItem( hWndDlg, IDC_STORAGE, &ci, &hMyVars, &lpMyVars );
        here.x=LOWORD( lParam );
        here.y=HIWORD( lParam );
        whichpoint=GraphHandleWM_LBUTTONDOWN( ci, lpMyVars->hGraph, here );
}
```

Returns: The index of the point that is under the mouse. A new point may have been created if the user clicked in an empty area of the graph. This index will always range from 0 to one minus the number of points in the graph.

## int GraphHandleWM_LBUTTONUP ( ci, hGraph, ptCursor ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |
| **POINT ptCursor** | Coordinate of mouse up point (gained from lParam) |

Call this function in response to the WM_LBUTTONUP message in the settings dialog. See GraphHandleWM_LBUTTONDOWN for more specifics.

Returns: The index of the point that is under the mouse.

## int GraphHandleWM_LBUTTONDBLCLK ( ci, hGraph, ptCursor ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |
| **POINT ptCursor** | Coordinate of mouse double click point (gained from lParam) |

Call this function in response to the WM_LBUTTONDBLCLK message in the settings dialog. See GraphHandleWM_LBUTTONDOWN for more specifics.

Returns: The index of the point that is under the mouse.

## int GraphHandleWM_MOUSEMOVE ( ci, hGraph, ptCursor ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |
| **POINT ptCursor** | Coordinate of mouse point (gained from lParam) |

Call this function in response to the WM_MOUSEMOVE message in the settings dialog. See GraphHandleWM_LBUTTONDOWN for more specifics.

Returns: The index of the point that is under the mouse.

## void GraphDestroy ( ci, hGraph ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hGraph** | Handle to graph given by **GraphCreate()** |

Call this function to destroy the graph. This is usually done in the **XfmDestroy()** function. After the graph is destroyed, the handle is no longer valid, and all memory used by the graph is freed.

## void GetTempName ( ci, szThree, szFilename ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **LPSTR szThree** | Three letter identifier for type of temporary file |
| **LPSTR szFilename** | Complete filename of temporary file |

This function returns the name of a valid unique temporary file name in szFilename. The user's choice of temporary directory in Cool Edit is used as the directory for this file.

## HANDLE CreateXfmVars ( ci, cszXfmName ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **LPCSTR cszXfmName** | Name of the other effects module |

Other effects modules can be used from within your effects module if they exist. For example, you can possibly use Cool Edit's Filter or Amplify functions from within your own. But since the internal structures for these functions may change over time, you are better off just using other XFM modules that you have created. When called, the external structure is initialized to its defaults (from the other module's XfmInit() function).

Returns: Handle to be used in subsequent calls for identifying the external effect module's data structure

## void SetXfmVar ( ci, hVars, iOffset, iLength, lpvMem ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **HANDLE hVars** | Handle for use with external transforms (from **CreateXfmVars()**) |
| **int iOffset** | Offset into transform's structure |
| **int iLength** | Length of data in transform's structure |
| **void far * lpvMem** | Memory to copy to transform's structure |

Set some data in another effect module's user defined data structure. After the data is set, the effect may be called using **CallXfm()**.

## long CallXfm ( ci, cszXfmName, hVars, dwLoSamp, dwHiSamp, bShowMeter ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **LPCSTR cszXfmName** | Name of the other effects module |
| **HANDLE hVars** | Handle for use with external transforms (from **CreateXfmVars()**) |
| **DWORD dwLoSamp** | Low sample value to work with |
| **DWORD dwHiSamp** | High sample value to work with |
| **BOOL bShowMeter** | Should the called effect show its progress meter? |

Actually call the other effect. The effect will work with samples from dwLoSamp on up to dwHiSamp. If your function is displaying a progress meter, you will want to set bShowMeter to FALSE so the effect you are calling doesn't try to display its own progress meter as well (which would end in total confusion and chaos).

Returns: Value depends on function being called. It is the return value from XfmDo() of the effect being called.

## void SetWindowType ( ci, wType ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **WORD wType** | Type of window |

Sets the window type when windowing data for an FFT (when using **WindowFFT()** and **IWindowFFT()**). Window types can be any one of the following:

| | |
|---|---|
| 0 | Triangular |
| 1 | Von Hann |
| 2 | Hamming |
| 3 | Blackman |
| 4 | Cosine Squared |
| 5 | Blackman-Harris |

The values go from narrowest band but widest sidelobes and gentle slopes to wider bands with smaller sidelobs and steeper slopes. In general, Hamming is used most often, but Blackman will give a little steeper slopes.

## void WindowFFT ( ci, lpData, iSize, bStereo ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **float far * lpData** | Pointer to FFT prepared data (prepare with the SetFFT... functions) |
| **int iSize** | Size of FFT (number of data points) |
| **BOOL bStereo** | TRUE for stereo, FALSE for mono |

Once data is converted from 16-bit or 8-bit audio to floating point data, the floating point data can be windowed before performing the FFT. Windowing the data consists of multiplying each sample by the window values. Samples near the center are multiplied by 1.0 while samples near the end are multiplied by smaller values down to but not including 0. Windowing the data will give a more precise measurement of the frequencies present, since the data is assumed to be circular. Please read about FFTs in other material related to Digital Signal Processing for more information.

## void IWindowFFT ( ci, lpData, iSize, bStereo ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **float far * lpData** | Pointer to FFT prepared data (prepare with the SetFFT... functions) |
| **int iSize** | Size of FFT (number of data points) |
| **BOOL bStereo** | TRUE for stereo, FALSE for mono |

Perform the inverse window on FFT data. Callint IWindowFFT() right after WindowFFT will result in no change in the original data. Instead of multiplying the data by the window coefficients, the data is divided by the window coefficients.

## void FFT ( ci, lpData, iSize, iDirection ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **float far * lpData** | Pointer to FFT prepared data (prepare with the SetFFT... functions) |
| **int iSize** | Size of FFT (number of data points) |
| **int iDirection** | 1 for FFT, -1 for inverse FFT |

Perform an FFT (Fast Fourier Transform) on the prepared data. The floating point values will then contain pairs of {real,imaginery} data starting at index 1, so lpData[1] contains the real part of the first bin, and lpData[2] contains the imaginery part of the first bin, and so on. The data going in to the FFT should have been prepared from the original audio data using one of the SetFFT... functions. The largest size of FFT that should be used with this function is a 4096 point FFT, so iSize should never be more than 4096, and must be a power of 2.

After the FFT is performed, the data can be manipulated, or viewed, or whatever. If it is manipulated in some way (eg changing the phase or amplituded of various frequency components), then the inverse FFT can be performed by setting iDirection to -1, then an inverse Window performed by calling IWindowFFT(), and finaly calling the GetFFT... function to convert the floating point data back to audio data. Please read books on Digital Signal Processing for more information on the FFT.

## void LFFT ( ci, hpReData, hpImData, lSize, iDirection ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **double huge * hpReData** | Pointer to array of lSize doubles representing the Real components |
| **double huge * hpImData** | Pointer to array of lSize doubles representing the Imaginery components |
| **long iSize** | Size of FFT (number of data points) |
| **int iDirection** | 1 for FFT, -1 for inverse FFT |

Perform an FFT (Fast Fourier Transform) on the data. The largest sized FFT supported depends on the amount of RAM the user has, which will be evident when the huge arrays are allocated by your DLL. On the average, FFT sizes up to 262,144 points can be performed successfully on most systems. The data is zero-based, so hpReData[0] is the first real value part of the first data point for example. The number of data points should only have the factors 2,3,4, and 5. So an FFT of size 10,000 is fine, since that 10,000 factors to 5 * 5 * 5 * 5 * 4 * 4. So, any powers of 2, 3, 4, or 5 are also valid by this rule.

After the FFT is performed, the data can be manipulated, or viewed, or whatever. If it is manipulated in some way (eg changing the phase or amplituded of various frequency components), then the inverse FFT can be performed by setting iDirection to -1. You must provide any windowing necessary on this data, as the only "Large" FFT function provided is this one.

## void SetFFT16bit ( ci, lpFloats, lpiAudio, iSize )
## void SetFFT8bit ( ci, lpFloats, lpcAudio, iSize ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **float far *lpFloats** | Destination array of floating point values for use with FFT functions |
| **int far * lpiAudio** | Source array of 16-bit samples |
| **usigned char far * lpcAudio** | Source array of 8-bit samples |
| **int iSize** | Number of samples to prepare/copy. |

Fill an array of complex floats with integer audio data. There will be twice as many floating point numbers generated as iSize since complex numbers require two floats (one for the real part, and one for the imaginery part). The floats array must be large enough to hold *iSize*2+1* floating point values (the +1 is because the array is 1 based, not zero based). iSize integer samples from lpiAudio will be converted to iSize complex floating point values in lpFloats.

If preparing 8-bit audio for an FFT, the SetFFT8bit function is used, and character data is read instead of integer data.

## void SetStereoFFT16bit ( ci, lpFloats, lpiLeft, lpiRight, iSize )
## void SetStereoFFT8bit ( ci, lpFloats, lpcLeft, lpcRight, iSize ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **float far *lpFloats** | Destination array of floating point values for use with FFT functions |
| **int far * lpiLeft** | Source array of 16-bit samples for left chananel |
| **int far * lpiRight** | Source array of 16-bit samples for right channel |
| **usigned char far * lpcLeft** | Source array of 8-bit samples for left chananel |
| **unsigned char far * lpcRight** | Source array of 8-bit samples for right channel |
| **int iSize** | Number of samples to prepare/copy. |

Two channels can be converted to one set of floating point data at a time for more efficiency. This way, two FFTs can be performed in parallel.

## void SetStereoFFT16bitInterleaved ( ci, lpFloats, lpiAudio, iSize )
## void SetStereoFFT8bitInterleaved ( ci, lpFloats, lpcAudio, iSize ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **float far *lpFloats** | Destination array of floating point values for use with FFT functions |
| **int far * lpiAudio** | Source array of 16-bit samples, left/right interleaved |
| **usigned char far * lpcAudio** | Source array of 8-bit samples, left/right interleaved |
| **int iSize** | Number of samples to prepare/copy. |

Two channels can be converted to one set of floating point data at a time for more efficiency. This way, two FFTs can be performed in parallel. The source data is interleaved left sample 0, right sample 0, left sample 1, right sample 1, ...

## void GetStereoFFT16bitInterleaved ( ci, lpFloats, lpiAudio, iSize, iOp)
## void GetStereoFFT8bitInterleaved ( ci, lpFloats, lpcAudio, iSize, iOp ) COOLINFO function

| | |
|---|---|
| **COOLINFO FAR *ci** | Cool Edit info structure pointer |
| **float far *lpFloats** | Source array of floating point values for use with FFT functions |
| **int far * lpiAudio** | Destination array of 16-bit samples, left/right interleaved |
| **usigned char far * lpcAudio** | Destination array of 8-bit samples, left/right interleaved |
| **int iSize** | Number of samples to prepare/copy. |
| **int iOp** | Operation - set this to zero. |

Convert floating point FFT data (data gained after doing an inverse FFT) back to interleaved audio.

To convert FFT data back that was set using SetFFT16bit, simply convert every other floating point value starting with offset 1. So IntegerSample[0] = (int)lpFloats[1], IntegerSample[1] = (int)lpFloats[3], IntegerSample[2] = (int)lpFloats[5] and so on until you have IntegerSamploe[iSize-1] = (int)lpFloats[(iSize-1)*2+1]

## Definitions

```
#define XFM_MONO8          1
#define XFM_STEREO8        2
#define XFM_MONO16         4
#define XFM_STEREO16       8

#define XFM_HELPFILE "cool.hlp"   // Put your own help file here if you support the [?] Help button

typedef DWORD (CALLBACK*     DWFARPROC)();            // Used for COOLINFO structure functions
typedef void  (CALLBACK*     VFARPROC)();             // Used for COOLINFO structure functions

typedef struct xfmquery_tag
{           char szName[40];             // Appears in Menu (without elipsis...), eg "Echo" will create the menu item "Echo..."
            char szCopyright[80];        // Appears in the progress meter dialog when processing
            char szToolHelp[70];                 // Appears in the quick-help below toolbar icon
            WORD wSupports;              // OR's of XFM_MONO16, XFM_STEREO16, XFM_MONO8, XFM_STEREO8  A
                                         // function only needs to support MONO and STEREO 16-bit, Cool Edit will do the
                                         // necessary conversions to work with 8-bit data.
            DWORD dwFlags;               // Any of the XF_ flags OR'd together
                                         // Must at least have either XF_TRANSFORM, XF_GENERATE or XF_ANALYZE
            DWORD dwUserDataLength;      // Length of transform's internal data
            char szStructDef[48];        // Array of chars representing types in user data area
            char szPresetDef[48];        // Array of chars 'y' to include in presets, 'n' to ignore.
} XFMQUERY;

#define XFM_VALIDLIBRARY             150       // This stands for the earliest version of Cool Edit that this API supports

#define XF_TRANSFORM                 1         // This function transforms highlighted audio
#define XF_GENERATE                  2         // This function generates new pcm wave data
#define XF_ANALYZE                   1024      // This function analyzes the highlighted audio data

#define XF_USESPRESETSAPI            4         // Has a presets box (and uses the presets/scripts API)
#define XF_USESGRAPHAPI              8         // Uses the built in graph control functions
#define XF_USESFFTAPI                256       // Uses Cool's FFT functions

#define XF_MUSTHIGHLIGHT             16        // If set, function only enabled if a selection is highlighted (normal)
#define XF_MUSTHAVECLIP              32        // If set, function is grayed if nothing on clipboard
#define XF_MODIFIESTOENDOFVIEW       64        // Set this if function modifies data outside user's given selection
#define XF_MODIFIESTOENDOFFILE       128       // Set this if function modifies data outside user's given selection
#define XF_NOSINGLEEDIT              512       // Do not include if editing only one channel

// See the text for in depth explanations of the various functions your XFM DLL can call.
typedef struct coolinfo_tag
{       WORD wChannels;                  // Number of channels
        WORD wBitsPerSample;             // Bit size, 8 or 16 for now
        WORD wBlockAlign;                // Bytes per sample (eg stereo 16-bit = 4)
        long lSamprate;                  // Sample rate (8000,11025,22050, etc.)
        HANDLE hUserData;                // Handle to specialized transform data, depends on DLL
        DWORD dwLoSample;                // Lowest sample to transform
        DWORD dwHiSample;                // Highest sample
        FARPROC lpTestFunction;          // Not Used
        XFMQUERY FAR *cq;                // Pointer to query struct

        // General Purpose Tools
        FARPROC lpCenterDialog;          // (HWND hWndDlg, int iUnused)  Center the dialog box

        // Reading and Writing audio data
        FARPROC lpReadData;              // (char huge *data,long offset,long amount)
        FARPROC lpWriteData;             // (char huge *data, DWORD offset,DWORD nbytes)

        // Progress Meter
        BOOL FAR *lpProgressCanceled;    // if TRUE, user hit Cancel button, you must stop processing NOW
        FARPROC lpProgressMeter;         // (DWORD dwCurrent, DWORD dwTotal)  percent done is 100*dwCurrent/dwTotal
        FARPROC lpProgressCreate;        // (LPCSTR szText)  szText is message to indicate type of processing
        FARPROC lpProgressDestroy;       // (void)  Call to remove progress meter
```

## Definitions continued

```
        // Presets
        char far * szIniFile;                   // Ini file to save preset info into
        FARPROC lpPresetsInit;
        FARPROC lpHandleID_PRESETNAME;
        FARPROC lpHandleID_PRESETS;
        FARPROC lpHandleID_ADD;
        FARPROC lpHandleID_DEL;

        // Graph control
        FARPROC lpGraphCreate;
        FARPROC lpGraphCount;
        FARPROC lpGraphGetPoint;
        FARPROC lpGraphGetValueAt;
        FARPROC lpGraphDraw;
        FARPROC lpGraphClear;
        FARPROC lpGraphInverse;
        FARPROC lpGraphDestroy;
        FARPROC lpGraphHandleWM_LBUTTONDOWN;
        FARPROC lpGraphHandleWM_LBUTTONUP;
        FARPROC lpGraphHandleWM_LBUTTONDBLCLK;
        FARPROC lpGraphHandleWM_MOUSEMOVE;
        FARPROC lpGraphSetDblClkScales;
        FARPROC lpGraphSetDblClkNames;
        FARPROC lpGraphSetDialog;

        int iScriptFile;
        int iScriptDialogStop;
        HFONT hFont;                            // smaller font for dialogs

        // for XF_GENERATE type, must be filled out during XfmSetup
        BOOL bReplacesHighlightedSelection;     // if TRUE, highlighted selection is deleted before inserting blanks
        DWORD dwInsertBlankSamples;             // the number of samples to generate
        BOOL bHasCoprocessor;                   // if TRUE, use fp, otherwise use table lookups and other speedups
        FARPROC lpFFT;                          // perform n-point FFT (radix-2 for now)
        FARPROC lpWindowFFT;
        FARPROC lpIWindowFFT;
        FARPROC lpSetWindowType;                // 0..5: Triangular, von Hann, Hamming, Blackman, Cosine^2, Blackman-Harris
        FARPROC lpGetTempName;                  // get temporary file name

        FARPROC lpSetFFT16bit;
        FARPROC lpSetFFT8bit;
        FARPROC lpSetStereoFFT16bit;
        FARPROC lpSetStereoFFT8bit;
        FARPROC lpSetStereoFFT16bitInterleaved;
        FARPROC lpSetStereoFFT8bitInterleaved;
        FARPROC lpGetStereoFFT16bitInterleaved;
        FARPROC lpGetStereoFFT8bitInterleaved;

        FARPROC lpCreateXfmVars;
        FARPROC lpSetXfmVar;
        DWFARPROC lpCallXfm;

        // Delayed writing
        VFARPROC lpDelayWrite;
        VFARPROC lpDelayWriteInit;
        VFARPROC lpDelayWriteDestroy;

        HINSTANCE hInst;                        // This is the instance of your DLL

        FARPROC lpGraphCopy;
        FARPROC lpCutData;
        FARPROC lpInsertBlankData;
        FARPROC lpDeleteXfmVars;

        DWORD dwRightSample;                    // Current viewing screen rightmost sample
        DWORD dwLeftSample;                     // and leftmost sample
        FARPROC lpGraphSetPoint;

        DWORD dwExtraFlags;
        FARPROC lpLFFT;
} COOLINFO;
```

## Appendix A - Sample File Filter (au.flt)

If you would lilke the sample code for the AU.FLT file fitler, please send a request.

## Appendix B - Sample Transform Effects Module (distort.xfm)

If you would lilke the sample code for the DISTORT.XFM file fitler, please send a request.

## Appendix C - Sample Generate Effects Module (dtmf.xfm)

If you would lilke the sample code for the DTMF.XFM file fitler, please send a request.