

Help Index for SoundTool Version 3.1

Informations for Users:



[Info about SoundTool](#)



[What's new in this release ?](#)



[Getting started with SoundTool](#)



[The SoundTool window](#)



[The SoundTool menu](#)



[The SoundTool Command Line](#)



[Using Drag and Drop with SoundTool](#)



[Installing additional I/O libraries](#)

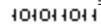


[Using Dynamic Data Exchange](#)

Informations for Programmers:



[Entries in SNDTOOL.INI](#)



[File formats](#)



[Clipboard usage](#)

[Examples for clipboard data exchange](#)



[Adding a DLL for playing sounds](#)



[Adding a DLL for recording sounds](#)

Info about SoundTool




for Microsoft Windows Version 3

© 1990-1992 by Martin Hepperle

SoundTool is a simple utility to manipulate sampled 8-bit mono sound data.

The included sound player relies on the dynamic link library DSOUND.DLL (© 1990-1991 by Aaron Wallace) to play a sound sample. This works under Windows 3.0 and Windows 3.1. To record samples you need either an AD-board or one of the sound boards (support for sound boards is not included in this free distribution). You can extend the voice I/O capabilities of SoundTool by writing your own libraries.

SoundTool can cut, copy and paste parts of a sample to the clipboard and perform various modifications to the whole sound sample or to a part of the sample. Recent enhancements include a DDE interface to provide a basis for voice mail and inclusion of voice data into your documents.

To use SoundTool efficiently a  is very helpful, but you can access all functions without such a beast too.

SoundTool is Shareware.

You should send a minimum of \$20.- or DM 30.- to the following address to keep your nights peaceful and to receive the latest version of SoundTool which **includes additional drivers** which record and play using your SoundBlaster board (under Windows 3.0 only) and drivers which use the multimedia(MM) extensions of Windows 3.1 to record and play on all devices (including SoundBlaster) which are supported by drivers of the MM-extensions.

You can send me an US Postal Money Order too, I can clear this without additional fees. At the present time it costs about \$5 to clear a check drawn on an US bank, so I don't see any other method that makes sense than sending money orders or banknotes in a letter (or I have to push the registration fee to \$25 for check lovers).

-- German users can use a bank transfer to my Bank account 1987 45-701 at Postgiroamt Stuttgart (BLZ 600 100 70).

I do not warrant that this software will meet your requirements or that this software will be error free. In no event will I be liable to the user of this software for any damage, including lost profits, lost savings or other incidental or consequential damages.

**Martin Hepperle
Robert-Leicht-Straße 175
D-7000 Stuttgart 80
Germany**

- This address will be valid until December 1992 -



Thank you for reading and have a nice day.

What's new in this version of SoundTool ?

This version of SoundTool looks like the previous one, but behind the scenes there are many differences.

New DLL format: (2.3)

The most important part is the complete extraction of recorder and player routines into dynamic link libraries. This makes it possible to use SoundTool with most sound board if you can supply the necessary driver.

Some manufacturers like Creative Labs (maker of the widely used Sound Blaster board) are supplying drivers to use their hardware under Windows which makes it quite easy to write the additional interface library to interface with SoundTool's I/O functions.

Registered users of SoundTool will receive the necessary libraries for the SoundBlaster, included in this release is the output library for the internal PC speaker and the input library for a generic D/A board which is available here in Germany.

New Ribbon: (2.4)

A new feature is the added ribbon (call me Toolbar) at the bottom of the SoundTool window which gives you quick access to some of the most often used functions without having to resort to the pulldown menus.

New DDE Interface: (2.4)

You can use DDE commands to record and sample sounds.

New Filter option: (2.4)

This makes it possible to filter a selected frequency band from a sound sample. You can eliminate background noise and change the sound of human voices. Works not too perfect yet.

Command line switches: (2.5)

The option `-b` makes it possible to play an audiofile without showing SoundTool. by using the switch `-x` you can ply a sample while starting other applications like WinWord or Excel (only interesting if you have installed a background player e.g. for the SoundBlaster).

Audio file format cleaned up: (2.5)

Now that I have got my SPARCstation, I was able to get a better view of the audio file format which is identical for the SUN as well as on the NeXT. SoundTool now supports linear 16-bit samples and μ Law encoded 8-bit samples.

If I could get some example files of the other audio file flavours I would be able to integrate them too.

User can customize frequency combobox: (2.5)

It is possible to add/remove entries in the frequency select box. The frequencies are stored in SNDTOOL.INI.

User can delete sound files: (2.5)

The File menu contains a new entry Delete... to delete a selected file.

Wave file format added: (2.5)

SoundTool now supports input of linear PCM-encoded 8-bit samples in Microsoft Multimedia RIFF/WAVE file format.

Examine unknown sound files: (2.5)

In the file open and file delete dialog boxes SoundTool can look at a file and tell you about the possible sound file format. It does this by examining the file header; it performs no

statistics about signed/unsigned or short/long samples.

DLL format extended: (2.5)

SoundTool now supports aborting playback and recording if you use a background player/recorder. The SLL libraries have been extended accordingly.

Drivers for multimedia extensions: (2.6)

SoundTool now supports recording and playback of sounds through two new SLL libraries which use the wave functions of multimedia extensions included in Windows 3.1.

Corrected features in file save procedures: (2.6)

SoundTool now really writes out nothing but the selected part of the sample (as stated in the docs).

New filter options: (2.6)

Two new averaging filter commands can remove background hiss.

Support for Drag and Drop: (2.7)

Instead of using the menu option File-Open you can drag files from filemanager to SoundTools window or icon to load them. You must have Windows 3.1 to use this function.

More informative WAVE files : (2.7)

When you save a file in WAVE format the description of the sample is embedded; when a file is read, the description is read back again.

VOC file output support added: (2.8)

SoundTool now can write files in Electronic Arts VOC file format.

Signed/Unsigned Raw files: (2.8)

When reading 8-Bit samples the user can select a sample representation either by signed or unsigned byte values.

Wave Generator: (2.9)

You can create sine-, square- or sawtooth. shaped waves of arbitrary frequency and duration.

Function Generator: (3.0)

Arbitrary shaped waves can be generated by entering a mathematical expression.

TouchTone: (3.1)

This function makes it possible to create the tones of a phone (touch tone dialer).

Drag and Play: (3.1)

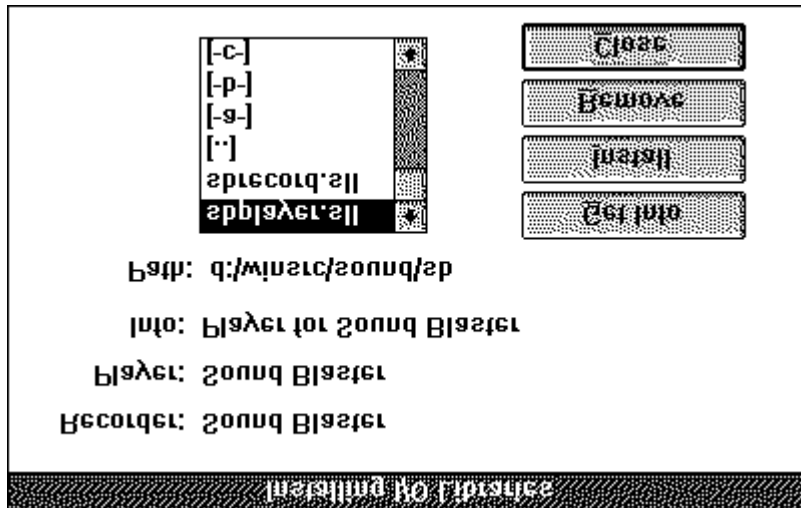
sound files can be dragged from the Windows file manager to SoundTools *icon*. SoundTool then loads, plays and removes (from memory) these samples. If SoundTool is displayed in its *regular size*, the files are loaded, but not played. You must have Windows 3.1 to use this function.

Getting started with SoundTool

If you have never used SoundTool before, you have to select the libraries for voice output and/or input.

To do this, follow the next steps:

- Select the menu command **File Install...**
- The following dialog box pops up:



- select the library you are interested in and press **[Get Info]**. A short description will appear to the right of the label Info:. To install this library press **[Install]**. You should install libraries only if they are written for SoundTool and if you have the necessary hardware.
- After you have installed the libraries close the Box by pushing the **[Close]** button.
- You can remove (un-install) a library by selecting the sll file and pushing the **[Remove]** button (Hint: it is not necessary to select exactly the same library, just the library *type* is important so that SoundTool can decide which library you want to remove (Player or Recorder)).

Using dragn drop with SoundTool

There are two ways to use drag and drop SoundTool:

- a) Drag and load:
If you drag a sound file from the Windows FileManager on SoundTool while SoundTool is shown in its **full size**, this sound file gets loaded. This function works with more than one file too.
- b) You can drag a sound file over the SoundTool **icon**. This will load the dropped file into SoundTool, play the file and remove it from SoundTools memory.

The SoundTool command line

There are several ways to use SoundTool with and without command line options:

- a) Install SoundTool into a group of the Windows **Program Manager**. Then you can start SoundTool by doubleclicking on the icon. This brings up SoundTool without any sound file loaded.
- b) Open the Windows **File Manager** so that you see SNDTOOL.EXE and doubleclick on that line. This brings up SoundTool without any sound file loaded. You can drag a file with extension SOU, SND, SNP, TXT, AU or NXT onto the line SNDTOOL.EXE and drop it there. This starts up SoundTool and loads the dropped file into SoundTool. -This will work only if SNDTOOL.EXE and the sound file reside in the same directory.

command line switch -

You can doubleclick onto a line containing a file with extension SOU, SND, SNP, TXT, AU or NXT. This starts up SoundTool with an invisible Window if you made a connection between SNDTOOL.EXE - and e.g. *.SND files (There is a space character followed by a minus sign behind SNDTOOL.EXE; you can execute SNDTOOL.EXE - SNDFILE.SND manually too). The Sound is loaded, played once and SoundTool removes itself from memory (This works the same way Aaron Wallaces SOUNDER.EXE does, but does distinguish between the different sound file formats). If you choose a file format which needs translation (like a SUN audio file) there will be a noticeable delay until the sound is actually played; - be patient !

Example:

```
SNDTOOL.EXE - HIDDEN.SND
```

command line switch -x

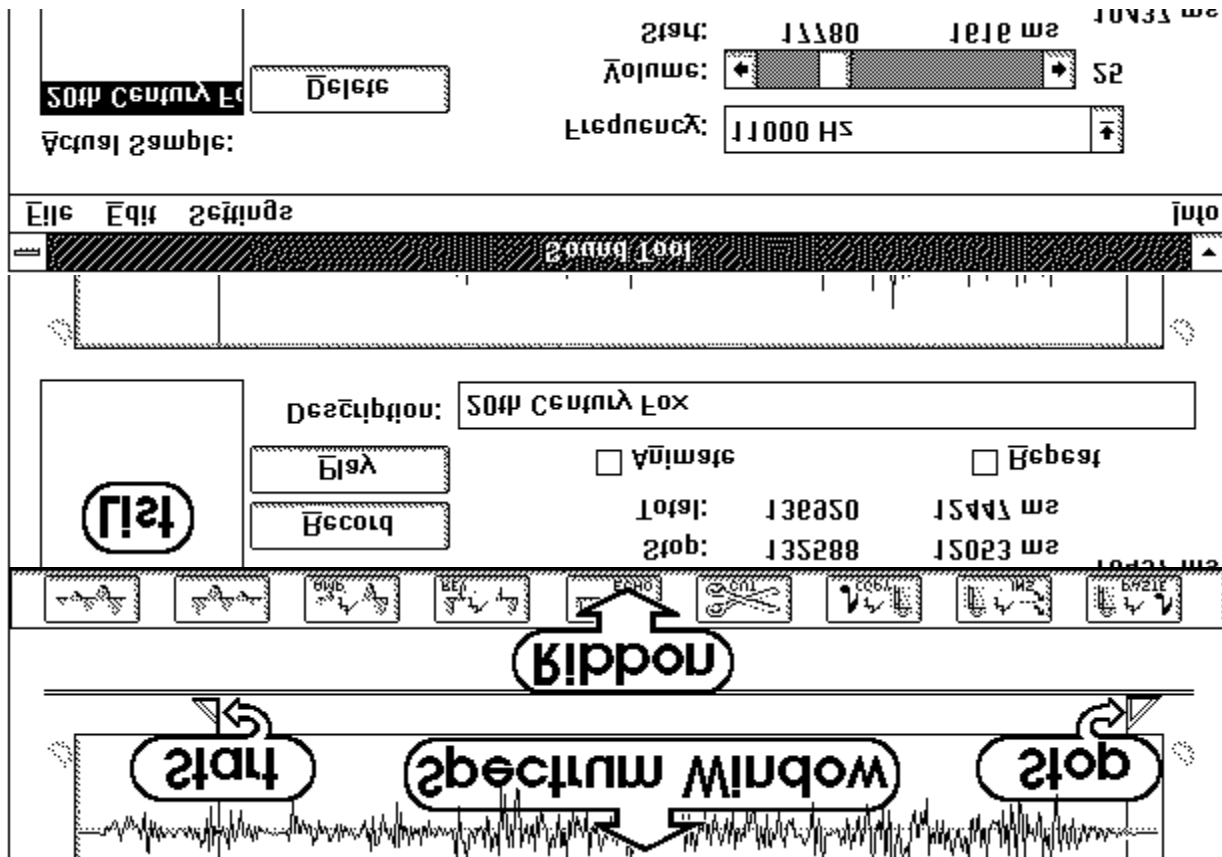
Immediately following the switch -x you can specify the name of an executable applikation (enclosed between double quotes). If you specify an additional name of an audio file which SoundTool can read, this sample is played and then the application following the -x gets started; if you are using a background player, the sound is playing *while* the application is started. To make sure that the sample is played at the correct speed, you should use a file format which contains frequency information (i.e. *not* a raw .SOU file).

Example:

```
SNDTOOL.EXE -x"D:\EXCEL\EXCEL.EXE" FANFARE.SND
```


The SoundTool window

After firing up SoundTool the following window is displayed:



Items in SoundTools window:

The window contains the following controls and displays:

Available Samples a listbox which displays a list of loaded sounds. You can select one to be the current sound by clicking with the mouse on it. If the samples are in SoundTool's *.SND format the listbox shows the description of the sample (at least the leftmost part of it), in all other cases the file name is shown.

Delete delete the current sound from memory by clicking with the mouse on this button. The sample is lost if you don't save it into a file before you select this option!

Record You can record a new sound by clicking on this button with the mouse. Starts sampling of Sound data. (Only applicable if you have an A/D-board and a matching Sound-Link-Library (SLL) installed). The sampling parameters can be set by the menu option Settings-Recorder.... You can abort the process by pressing the [Escape] key on your keyboard (not if you are using DSOUND.DLL). After a sample has been recorded, SoundTool removes leading and trailing silence from the data (values between +5 and -5 from the

range of -127 to +128 are considered to be silence). If you have no audio signal connected to your sampler, the complete sample may be empty. In this case SoundTool will not add the sample to its list.

Play

to play the selected part of the current sound you have to push this button. (Only applicable if you have an D/A-board and a matching Sound-Link-Library (SLL) installed). The player parameters can be set by the menu option Settings-Player.... You can abort the playback by pressing the [Escape] key on your keyboard (not if you are using DSOUND.DLL).

Animate

When this checkbox is checked, pressing the Play-button replays the sound and, while playing, the played part is colored in red; you can stop playing by pressing any key if you are using a foreground player library (like speaker.sll). This will set the Stop slider to the position where you pressed the key. Because the sample is played in small chunks corresponding to one pixel in the spectrum window, you will get a bad sound quality; --how bad it is, depends on the length of your sample. If you have installed a background player you cannot interrupt the playback, but you can move the sliders while the sample is played back. Animate is a helpful tool to locate a specific location in a sample by listening and pressing a key.

Repeat

When this checkbox is checked, pressing the Play-button replays the sound forever; you can stop playing by pressing and holding the [Esc]ape-key or by clicking the checkbox again while sound is playing. If you are using a foreground player, you should not press any other keys or try to switch to another application while the sound is playing; you could find yourself caught in an endless loop.

Frequency

Select the frequency which is used to play the marked region of the sound by selecting the appropriate entry in the combo box. You can add and remove custom frequency values by selecting the menu Options-Frequencies.

Volume

Change the sound volume by dragging the elevator of this slider. Some devices do not support volume control, you can change the volume of a sample by amplifying it. (Remark: the SoundBlaster libraries do not support volume control (at least on the SoundBlaster version 1.5; you must amplify the sample if you need a louder or softer volume).

Start-Slider

This is the left indicator in the spectrum window. It is used to indicate the first byte of a selection. You can change its position by pressing the mouse button in the left-pointing triangle beneath the spectrum window or near the left of the vertical dotted line in the spectrum display. Drag the slider into the position you desire and release the mouse button. You can use the keyboard to move the slider too, just move the focus to the triangle by clicking the mouse cursor on it or by pressing the [TAB] key until the triangular handle blinks. Then press and hold the [<] and [>] arrow keys until the numeric display shows the desired position. This will move the sliders in small increments --to move faster, hold the [Ctrl] key while pressing the direction keys or press the [Home] or [End] keys to place the slider at the leftmost or

rightmost possible position.

There is a label **Start** below the volume slider which shows the starting byte count and the start time relative to the beginning of the sample.

Stop-Slider

This is an indicator for the last selected byte; you can move it using the mouse or the keyboard in the same way as described above. There is a label **Stop** below the **Start** label which indicates the last byte to be played and the time relative to the beginning of the sample.

Between the two label lines to the right the difference of start and stop time is displayed. The total length of the sample is displayed below these lines.

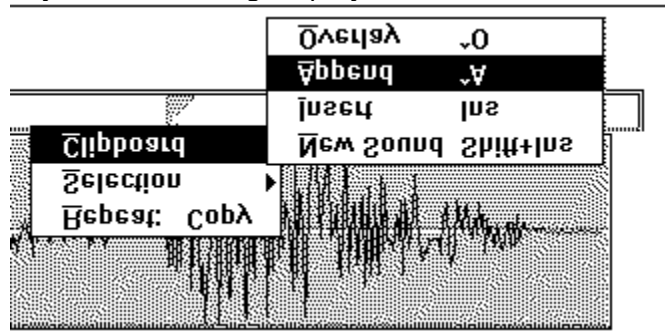
Description

This text string describes the sound; it may be up to 95 characters. It is saved to disk if you use one of SoundTools file formats (SND or SNP).

Spectrum window

This display shows the spectrum of current sound. The two sliders near the bottom (**Start** and **Stop**) can be moved by clicking on them and moving the mouse or by using the keyboard (see above *Start- and StopSlider*).

The left slider indicates the start of a selection, the right one marks the end of the selection. You can have quick access to the Edit submenu by clicking the right mouse button in the spectrum window or by pressing the [Return] key while the spectrum window is active (move the focus by pressing the [TAB]-key). Select the menuitem as usual by clicking the left mouse button or by moving the direction keys. The resulting display is shown below:

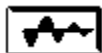


Ribbon

This tool ribbon gives you access to some of the most often used menu entries:



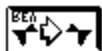
Fade In selection



Fade Out selection



Amplify selection



Reverse selection



Echo selection



Cut selection to clipboard



Copy selection to clipboard



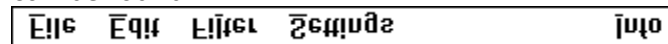
Insert clipboard into current sample (at Start position)



Paste clipboard to new sample

Menu Items:

SoundTool displays a menu bar near the top of it's window where the following menu items can be found:



File

⇒ File Open...

Loads the selected sound file into memory and updates the listbox.

It is possible to load one of the following sound file types:

⇒ **Raw 0...256** *.SOU 8-Bit *unsigned* sampled raw sound bytes without any header.

⇒ **Raw -127...128** *.SOU 8-Bit *signed* sampled raw sound bytes without any header.

⇒ **Sound** *.SND 8-Bit sampled sound bytes with header. These files can actually be in two formats:
1. files created by Aaron Wallace's SOUNDER including a short header.
2. files created by SoundTool including a longer header (see description of file formats below). SoundTool automatically detects which kind of SND file it is reading.

⇒ **Packed Sound** *.SNP 8-Bit sampled sound bytes compacted by a difference encoder (see file formats).

⇒ **NeXT** *.NXT see NeXT item:

⇒ **SUN Audio** *.AU Audio sound formats used by the NeXT computer and by SUNs SPARCstations. Currently these files must contain either 8-Bit samples in μ Law-format or linear 16-Bit samples which are reduced by SoundTool to 8-Bit linear.

⇒ **ANSI** *.TXT Text format. The file must contain integers in the range 0...255 separated by blanks, tabs or newlines. It is possible to create a text file of this structure e.g. with Microsoft Excel by specifying a function like '255*(sin(x)+1)'.
Example for a legal file format:

```
127 200 220
230
255
```

```
220
```

```
220
```

⇒ **Vocal** *.VOC sound format used by Creative Labs Soundblaster software. These files may contain 8-Bit samples which are read by SoundTool. Other data are ignored.

⇒ **IFF** *.IFF **Interchange File format**: sound format defined by Electronic Arts, used mainly on the Commodore Amiga platform (also known as AIFF format). SoundTool can read uncompressed 8-Bit samples only. Other data is ignored.

⇒ **WAVE** *.WAV **Microsoft WAVE format**. SoundTool can read 8-Bit PCM mono samples only.

When you have got a file of unknown file format, you can try to examine the file header by pushing the [Examine] button. SoundTool will tell you about the file format if it can identify the file.

⇒ **File Save...**

Writes the selected part of the current sound to a file. The various formats are explained above (File Open...).

⇒ **File Delete...**

Deletes a selected sound file from your disk after confirmation.

⇒ **File Install...**

Used to install additional Dynamic Link Libraries for sound I/O boards. These libraries use filenames with the extension .SLL.

Edit

(This menu is available to quick access in the spectrum window, see: [the SoundTool window](#))

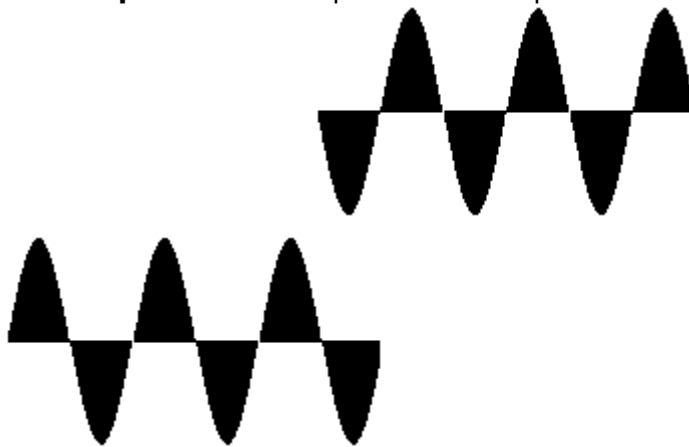
⇒ **Repeat last command**

This option repeats the last command which is often easier than the access through the popup menu tree

⇒ **Edit Selection**

⇒ **Flip vertical**

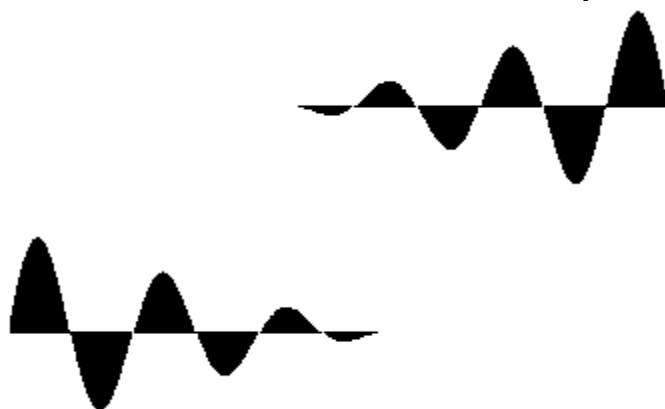
flips the marked part of the spectrum in vertical direction



gets flipped to:

⇒ **Reverse**

reverses the byte sequence of the marked part of the spectrum



gets reversed to:

⇒ **Expand to nnn%** expands/shrinks the marked part of the spectrum, uses linear interpolation on expansion to produce a good result.

This has the same effect as a local change of frequency to nnn% of the previous value.



When
is expanded to 200% the result looks like:



When it is expanded to 50% the result looks like:

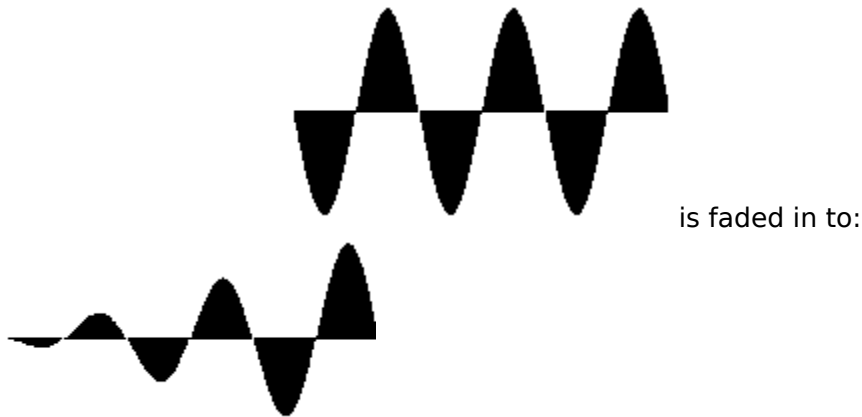
- ⇒ **Amplify to nnn%** scales each bytes in the selected area of the spectrum by a factor of $nnn/100$.
This results in a local in- or decrease of amplitude (sound volume). If the sound amplitude would exceed the limits (± 127) the result is clipped to these limits. The Amplification rate can be set by the menu option Settings-Amplification...



can be scaled to:



- ⇒ **Fade In** fades the bytes in the selected area of the spectrum by an linear increasing factor raising from 0 % to 100 % of the original value.
This results in a local fade in effect.



⇒ **Fade Out** fades the bytes in the selected area of the spectrum by a linear decreasing factor raising from 100 % to 0%.
This results in a local fade out effect.



⇒ **Echo** produces echos in the marked part of the spectrum, starting at the position of the left slider. If necessary the sample length is enlarged to avoid truncation of echos. The echo parameters can be set by the menu option Settings-Echo...

- ⇒ **Copy** copies the selected part of the spectrum to the clipboard
- ⇒ **Cut** cuts the marked part from the spectrum to the clipboard
- ⇒ **Delete** deletes the marked part from the spectrum
- ⇒ **Trim** trims the sample down to currently selected part

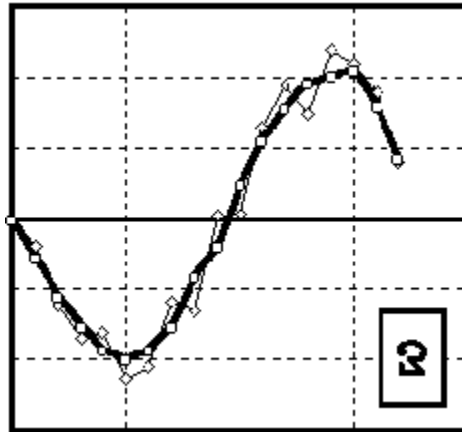
⇒ **Edit Clipboard**

- ⇒ **New Sound** pastes the clipboard contents into a new sound slot, updates the listbox
- ⇒ **Insert** inserts the clipboard contents at the position marked by the left slider in the spectrum window
- ⇒ **Append** appends the clipboard contents to the end of the current spectrum
- ⇒ **Overlay** overlays the clipboard contents beginning at the position marked by the left slider in the spectrum window; useful for echo effects. If necessary the overlay is clipped to the length of the current sample.

Filter

- ⇒ **Average of 3** The selection is replaced by the result of an averaging procedure which calculates the average of three immediately following samples.

The three samples are centered around the resulting sample:



The thick line (squares) shows the result of applying Average of 3 to the samples represented by the thin line (diamonds). This operation results in a smoothed signal, small disturbances (hiss) are smoothed out, but the sample sounds a bit muffled.

⇒ **Average of 5** The same as Average of 3, but 5 values are used instead of three. The five samples are centered around the resulting sample. The smoothing is stronger than the Average of 3 filter.

⇒ **Filter...** specify the upper and lower bounds of the band you want to pass through the filter. If you want to see the frequency spectrum of the selection before it has been filtered, check the appropriate box. To start the filtering action select Edit-Selection-Apply filter

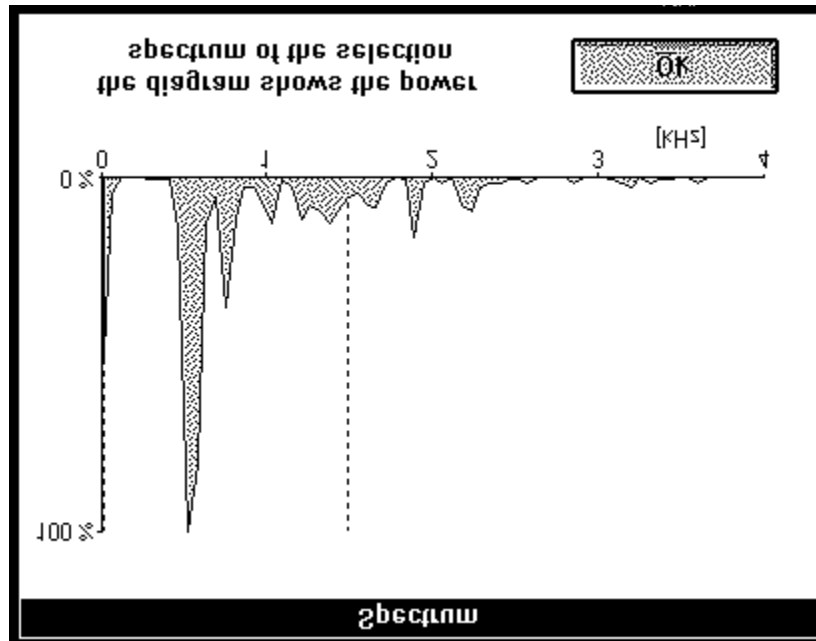
⇒ **Apply Filter**

applies the filter routine to the marked part of the spectrum. The part of the sample which passes the filter will replace the selection. You can specify the filter bandwidth under Settings-Filter....

SoundTool performs a *Fast Hartley Transform*, sets the coefficients of all frequencies falling outside the specified range to zero and then does an *Inverse Fast Hartley Transform* to create the new sample data. If you have set the lower bound to zero and the upper bound to the sample frequency or above no filtering will result, but you can get a look at the frequency spectrum of the selection (see: Settings-Filter... below). This filter option does not work perfectly well, it will be improved in later versions (I hope).

Hint: filtering can be quite time consuming, test this with a short selection of about 1000 or 2000 bytes first. You can abort the process by pressing the [Esc] key.

A typical result of a filter operation is shown below:



The filtered sample contains mostly frequencies in the range from 400 to 900 Hz, which results in two distinct peaks in this range.

The two dotted vertical lines near the left and near 1500 Hz show the filter bandwidth, everything outside this range will be suppressed by the filter if it would work optimal.

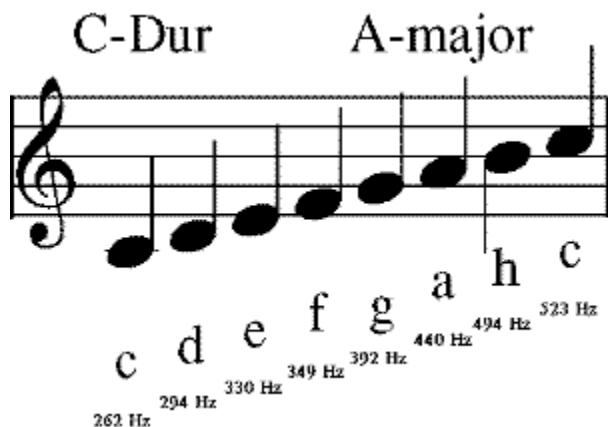
Generate

- ⇒ **Sine wave...** After you have entered the frequency and the duration SoundTool will generate a sample, using a sine waveform.
- ⇒ **Square wave...** same as above, but a rectangular waveform is used instead of a sine wave.
- ⇒ **Sawtooth wave...** same as above, but a sawtooth shaped waveform is used instead of a sine wave.

Note: A frequency of 440 Hz will give you an A of a minor in musical notation. This corresponds to the MIDI note 69.

The sampling frequency is fixed at 12000 Hz.

The following picture shows the frequencies of all notes in the first octave of A-major:



⇒ **Function wave...** After you have entered the number of samples **NS** and a valid expression SoundTool will evaluate the expression starting with **x=0** and incrementing **x** by **1** until **x=NS** is reached.
 The value of your expression at each **x** builds the resulting sample.
 The sampling frequency is fixed to 12000 Hz.

You want to try functions like:

$$\sin(x) \quad \text{or} \quad \sin(0.1*x) \quad \text{or} \quad \sin((0.01*x)^{1.5})$$

When defining your function, you should take into account, that the results should fall into the range from -1 to +1, larger resp. smaller values get clipped.

If you select *the Autoscale* option, SoundTool scales the sample so that it fits the range.

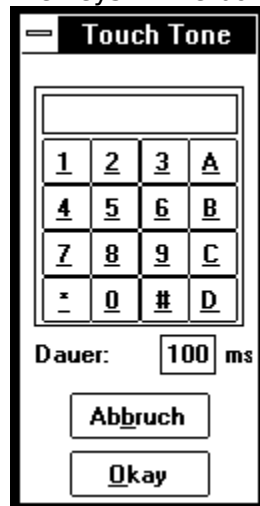
The following basic functions can be used to build your expression:

sin(x) cos(x) tan(x) sqrt(x) exp(x) ln(x) ^
 + - * /

⇒ **TouchTone...**

In many countries the local phone companies use a touch tone dialing system: each digit or letter in a phone number is represented by a unique combination of two frequencies.

SoundTool can create a sample from a given phone number. You can enter the number sequence in the dialogbox TouchTone by pressing the keys in the box or on your keyboard.



You can specify the duration per tone, this value must be at least 100 ms (1/10

Second).

Settings

⇒ **Echo...**

enter the number of echos and the delay between echos:
 If the delay between echos is too small you will get something that sounds like a direct feedback (computer voice).

⇒ **Amplification...**

enter the amplification factor which is used by the menu option Edit-Selection-Amplify to nnn%.

⇒ **Expansion...**

enter the expansion factor which is used by the menu option Edit-Selection-Exoand to nnn%.

⇒ **Clipboard...**

select the formats you wish to copy to the clipboard in addition to the standard format CF_SOUND.

You can render the selected part of the sample in text format (puts a

short description of the sample into the clipboard) and in metafile format (builds a complete representation of the selected part similar to the display in the spectrum window).

Warning: Metafile pictures can get quite large and take considerable amounts of processing time, because they are representing the complete sample, with one stroked line per byte.

- ⇒ **Options...** Offers the possibility to customize some options. This version allows to toggle the display of the toolbar.
- ⇒ **Frequencies...** pops up a dialogbox where the user can customize the values which are displayed in the frequency combobox. These values are stored in SNDTOOL.INI and loaded at startup time.
- ⇒ **Recorder...** enter the number of samples to record and adjust the recording frequency. (Only applicable if you have an A/D-board and the matching recorder SLL installed).
- ⇒ **Player...** enter additional parameters if your sound board needs them. (Only applicable if you have a D/A-board and the matching player SLL installed).

Info

- ⇒ **F1 Help** how the hell did you manage to display this help text ?
- ⇒ **Info...** same boring stuff as usual, but... wait a bit...

File formats

SoundTool can save sound samples in various formats:

8 bit raw format this is nothing more but a stream of bytes without any header.
ANSI format this is the same like the above raw format, but in a human readable form, one byte per line.

SUN Audio format,
NeXT Audio format: 8 bit samples encoded according to a μ Law rule or 16 bit linear samples.

These files are identified by the following header:

```
char szMagic[4] = { 'S', 'S', 'N', 'D' }
LONG_BE lStartOfData; /* offset into the file */
LONG_BE lDataSize;    /* length of data (optional) */
LONG_BE lEncoding;    /* 1 = 8-bit ISDN u-Law */
                        /* 3 = 16-bit linear PCM */
LONG_BE lSampleRate; /* frequency in [Hz] */
LONG_BE lChannels;    /* 1 = mono */
(LONG_BE typedefs 32 bit values in big endian (Motorola) format.)
```

Sounder SND: has been defined by Aaron Wallace and is used by his program Sounder. It contains a short header of 32 bytes of which 8 bytes are actually used:

```
WORD wSampleSize
WORD wFrequency
WORD wVolume
WORD wShift
```

SoundTool SND: contains some more informations in it's header:

```
char szMagic[6] = { 'S', 'O', 'U', 'N', 'D', 0x1a }
GLOBALHANDLE hGSound; /* currently not used, (prep. for
Stereo) */
DWORD dwBytes;        /* length of complete sample */
DWORD dwStart;        /* first byte to play from sample */
                        /* (0...dwBytes-1) */
DWORD dwStop;         /* first byte NOT to play from sample */
                        /*
                        /* (1...dwBytes) */
WORD wFreq;           /* frequency in Hz */
WORD wSampleSize;    /* see DSOUND.DLL from Aaron Wallace */
WORD wVolume;        /* 0...100 */
WORD wShift;         /* see DSOUND.DLL from Aaron Wallace */
char szName[96];     /* name of sound, ANSIZ */
```

The part of this header which follows the magical string is identical to the one used for clipboard transfer.

SoundTool SNP: contains the sample in a packed format:

```
char szMagic[6] = { 'S', 'N', 'D', 'P', 'K', 0x1a }
GLOBALHANDLE hGSound; /* currently not used, (prep. for
Stereo) */
DWORD dwBytes;        /* length of complete sample */
DWORD dwStart;        /* first byte to play from sample */
                        /* (0...dwBytes-1) */
DWORD dwStop;         /* first byte NOT to play from sample */
                        /*
                        /* (1...dwBytes) */
WORD wFreq;           /* frequency in Hz */
```

```
WORD wSampleSize;    /* see DSOUND.DLL from Aaron Wallace
*/
WORD wVolume;        /* 0...100 */
WORD wShift;         /* see DSOUND.DLL from Aaron Wallace
*/
char szName[96];     /* name of sound, ANSIZ */
(The only difference to SND is the szMagic number).
```

The compression method simply uses the difference between two succeeding bytes which is often smaller than ± 7 so that it can be stored in 3 bits. This simple and fast algorithm results in an average compression to 65 % of the original length (a Huffman compressor resulted in a reduction to 69% while needing three times the time for writing and the same time for reading, LHarc produced a file of 59 % and took the same triple time to process the data).

WAVE:

This format is used by the Multimedia-Extensions inside a RIFF file. SoundTool can handle linear 8-Bit PCM samples in mono quality. A complete description of this format can be found in Multimedia Programmers Reference, Microsoft Press.

Entries in SNDTOOL.INI

SoundTool uses SNDTOOL.INI to store some parameters under the caption [SoundTool]. Recorder and player libraries can store their data under the same application name if they avoid conflicts with SoundTool's key names.

[SoundTool]
; path name which was saved when Save path was checked in the File Open dialog box
path=d:\winsrc\sound\sun
; info about additional formats copied to the clipboard.
; This is the sum of the following values:
; 0 = Standard SoundTool sample.
; 1 = CF_TEXT, a short description of the sample
; 2 = CF_METAFILEPICT, a metafile picture of the sample
Clipboard=0
; toggles display of ribbon with buttons
; 0 = no
; 1 = yes
Ribbon=0
; toggles display of spectrum after filter option
; 0 = no
; 1 = yes
Spectrum=0
; number of last file format read or written
; numbers may change in later releases
FileType=...
; name and path of the player library if installed
PlayerDLL=D:\WINSRC\SOUND\SB\SBPLAYER.SLL
; name and path of the recorder library if installed
RecorderDLL=D:\WINSRC\SOUND\SB\SBRECORD.SLL
; Do you want to get audible feedback while opening and moving SoundTools windows ?
Noisy=1
; List of available frequencies (in ascending order)
Frequency0=4000 [Hz]
Frequency1=8000 [Hz]
Frequency2=11000 [Hz]
Frequency3=22000 [Hz]

The following key names are used by my recorder libraries:

; length of recorded sample
RecordBytes=1000000
; frequency at which we are sampling (may need additional adjustment)
RecordFrequency=11000
; used by btz13rec.sll:
; loop count to adjust the frequency
RecordDelay=192
; write to this port starts sampling of one byte.
RecordStartPort=0x0301
; the sampled byte can be read from this port:
RecordReadPort=0x0300

Clipboard data exchange structure

SoundTool registers a clipboard format "CF_SOUND" which can be used to exchange sound data between applications. Clipboard data in "CF_SOUND" format consists of a structure which contains general data, followed by the bytes which build the current sound spectrum.

The following data structure is used for clipboard transfer and inside SoundTool:

```
#define  DESC_LEN      96          /* max. length of a filename */
typedef struct sound_tag
{
    GLOBALHANDLE hGSound;        /* not used for clipboard transfer */
    DWORD dwBytes;              /* length of complete sample */
    DWORD dwStart;              /* first byte to play from sample */
    DWORD dwStop;               /* first byte NOT to play from sample */
    unsigned short usFreq;
    unsigned short usSampleSize;
    unsigned short usVolume;
    unsigned short usShift;
    char szName[DESC_LEN];      /* name of sound */
} SAMPLE;
```

usFreq must be one of the following values:
{ 5500, 7330, 11000, 22000 }

The other two data formats which are exported by SoundTool are the Windows standard formats CF_TEXT and CF_METAFILEPICT.

The CF_TEXT representation puts the following description into the clipboard:

```
Sound Sample
Description:  This sounds like a sample description
Length:      884 Bytes
Frequency:   22000 Hz
Volume:      20
Time:        19:26:19
Date:        09/28/91
```

The CF_METAFILEPICT format contains only MoveTo and LineTo segments which represent the sampled data. The amplitude values range from 0 to 2550, the time range is a 1:1 mapping of the sample bytes.

Examples for clipboard data exchange

The following two code fragments from soundtool show how to copy and paste sound data to/from the clipboard.

```

/*****
static SAMPLE Sound;
*****/
BOOL CopySound( HWND hWnd )
/* copy a sound sample to the clipboard */
{
    GLOBALHANDLE hGSample;
    SAMPLE FAR * lpSample;
    BYTE HUGE * lpCopySound;
    BYTE HUGE * lpSound;
    BOOL bReturn;
    DWORD dwBytes;

    bReturn = FALSE;
    dwBytes = min( Sound.dwBytes, (Sound.dwStop - Sound.dwStart) );
    if( NULL != (hGSample =
        GlobalAlloc( GMEM_MOVEABLE, sizeof(SAMPLE) + dwBytes ) ) )
    {
        if( NULL != (lpSample = (SAMPLE FAR *)GlobalLock( hGSample ) ) )
        {
            lpCopySound = sizeof(SAMPLE) + (BYTE HUGE *)lpSample;
            lpSound = (BYTE HUGE *)GlobalLock( Sound.hGSound );
            lpSound += Sound.dwStart;
            lpSample->dwBytes = dwBytes;
            lpSample->dwStart = 0;
            lpSample->dwStop = dwBytes;
            lpSample->usFreq = Sound.usFreq;
            lpSample->usSampleSize = Sound.usSampleSize;
            lpSample->usVolume = Sound.usVolume;
            lpSample->usShift = Sound.usShift;
            lstrcpy( lpSample->szName, Sound.szName);
            while( dwBytes-- )
            {
                *lpCopySound++ = *lpSound++;
            }
            GlobalUnlock( Sound.hGSound );
            GlobalUnlock( hGSample );

            if( OpenClipboard( hWnd ) )
            {
                EmptyClipboard();
                SetClipboardData( wFormat, hGSample );
                CloseClipboard();
                bReturn = TRUE;          /* yes, we finally did it ! */
            }
            else
            {
                /* cannot open clipboard, tell user about problem */
            }
        }
    }
    else

```

```

        {
            /* cannot lock sample structure, tell user about problem */
        }
    }
else
    {
        /* cannot allocate sample structure, tell user about problem */
    }

    return( bReturn);
}

/*****

BOOL PasteSound( HWND hWnd )
/* pastes sample from clipboard into next free slot */
{
    GLOBALHANDLE hGSample;
    SAMPLE FAR * lpSample;
    BYTE HUGE * lpCopySound;
    BYTE HUGE * lpSound;
    BOOL bReturn;
    DWORD dwBytes;

    bReturn = FALSE;

    if( wSounds >= MAXSOUNDS )
    {
        /* cannot paste any more sounds, tell user about problem */
        return( bReturn);
    }

    if( FALSE == OpenClipboard( hWnd ) )
    {
        /* cannot open clipboard, tell user about problem */
        return( bReturn);
    }

    if( NULL != (hGSample = GetClipboardData( wFormat )) )
    {
        if( NULL != (lpSample = (SAMPLE FAR *)GlobalLock( hGSample )) )
        {
            if( NULL != (Sound.hGSound =
                GlobalAlloc( GMEM_MOVEABLE, lpSample->dwBytes )) )
            {
                if( NULL != (lpSound =
                    (BYTE HUGE *)GlobalLock( Sound.hGSound )) )
                {
                    lpCopySound = sizeof(SAMPLE) + (BYTE HUGE *)lpSample;
                    Sound.dwBytes = lpSample->dwBytes;
                    Sound.dwStart = lpSample->dwStart;
                    Sound.dwStop = lpSample->dwStop;
                    Sound.usFreq = lpSample->usFreq;
                    Sound.usSampleSize = lpSample->usSampleSize;
                    Sound.usVolume = lpSample->usVolume;
                    Sound.usShift = lpSample->usShift;
                    lstrcpy( (Sound.szName), lpSample->szName );
                }
            }
        }
    }
}

```

```

        dwBytes = Sound.dwBytes;
        while( dwBytes-- )
        {
            *lpSound++ = *lpCopySound++;
        }
        GlobalUnlock( Sound.hGSound );
        bReturn = TRUE;    /* we finally arrived here */
    }
    else
    {
        /* cannot lock destination array, free it */
        /* and tell user about problem */
        GlobalFree( Sound.hGSound );
    }
}
else
{
    /* cannot alloc destination array, tell user about problem */
}
GlobalUnlock( hGSample );
}
else
{
    /* cannot lock clipboard structure, tell user about problem */
}
}
CloseClipboard();

return( bReturn );
}
/*****
/*                               end of sample code                               */
*****/

```

Adding a DLL for playing sound samples

SoundTool contains a mechanism that makes it possible for a Windows-programmer to incorporate player subroutines by writing a DLL which conforms to the following interface. Whenever SoundTool is started, it looks into [win.ini] to find an entry for a player library. These libraries are named with an SLL extension to avoid confusion (and possible crashes) with other DLLs when installing SoundTool libraries.

If an entry is found in SNDTOOL.INI, the corresponding library is loaded into memory and the menu of SoundTool offers an additional item Settings → Player... to call a setup procedure inside this DLL; if the library is not found the button Play is grayed and inoperative. To install a player library use the File → Install... menu command.

A player DLL must have at least seven exported functions with ordinal numbers @1 to @7 to make it usable with SoundTool.

- @1 WEP, the usual Windows Exit procedure required by every dynamic link library
- @2 This routine is called when the menuitem Settings → Player... is selected. The routine must conform to the following calling sequence:

```
BOOL FAR PASCAL PlaySetup( HWND );
```

The routine should ask the user for all the player parameters needed, and save them in the library data segment. The library is released when the user quits SoundTool, so it is a good idea to store needed Parameters in SNDTOOL.INI under the [SoundTool] caption. These parameters can be loaded when LibMain is called at load time of the library.

- @3 This routine is called when the button Play is pressed. The routine must conform to the following calling sequence:

```
void FAR PASCAL PlaySample( HWND, SAMPLE FAR * )
```

The first parameter is the handle of SoundTool's window, the second parameter points to a SAMPLE structure which contains all necessary information to play the sound. The routine should do nothing but play the data bytes which are addressed by the GLOBALHANDLE contained in the sample structure. The beginning and ending position should be taken from the structure. The SAMPLE pointer is valid only during processing this call (it may change during background playback if the user deletes sound entries), while the GLOBALHANDLE remains valid. The user cannot delete the data while it is played. When the routine finishes the playback it **must** post a message WM_PLAYDONE SoundTool's window; if you forget you will never leave SoundTool. If there is an error (e.g. device busy) the routine must post this message too, otherwise SoundTool will wait, and wait, and wait...

- @4 This routine must return LONG which contains a version number in the LOWORD and a magic word in the HIWORD (see example below). It is essential that the routine returns the magic number, otherwise SoundTool will refuse to load the library. To help the user to identify the library it must fill the string lpBuffer with an ANSI readable form of identification.

If the player is performing background transfers (returning immediately to SoundTool when playing a sample) the version number word must be Ored with 0x8000. Background transfer usually requires creating a window inside the DLL which is waiting for a message from the playing device driver and can be rather tricky. My SoundBlaster library does something like that.

The routine must conform to the following calling sequence:

```
LONG FAR PASCAL GetPLAYDLLVersion( LPSTR lpBuffer, int nMaxLength )
```

- @5** This procedure gets called, when the user presses the [ESC]-key. It returns a BOOL of TRUE if the playback cancelling was succesful. It is necessary to inform SoundTool about the end of playback by posting a message of WM_RECDONE too.
 BOOL FAR PASCAL StopPlayback(HWND)

- @6** Not yet used in version 2.6. Returns a BOOL of TRUE if the playback was successfully paused. SoundTool waits until the playing is canceled or continued.
 BOOL FAR PASCAL PausePlayback(HWND)

- @7** Not yet used in version 2.6. This procedure returns a BOOL which should be TRUE if the continuation of playback was successful.
 BOOL FAR PASCAL ContinuePlayback(HWND)

The following examples shows excerpts from my sample PLAYDLL which uses Aaron Wallace 's DSOUND.DLL. The library **must** contain at least the four exported ordinals.

PLAYDLL.DEF file showing EXPORTS with ordinal numbers.

```

LIBRARY    PLAYDLL
DESCRIPTION 'Player Library for use with SoundTool and IBM-PC speaker © Martin
            Hepperle 1991'

EXETYPE    WINDOWS

CODE       PRELOAD MOVEABLE DISCARDABLE
DATA       MOVEABLE SINGLE

HEAPSIZE   1024

EXPORTS
    WEP                @1 RESIDENTNAME
    PlaySetup          @2      ;necessary for SoundTool
    PlaySample         @3      ;necessary for SoundTool
    GetPLAYDLLVersion @4      ;necessary for SoundTool
    StopPlayback       @5      ;necessary for SoundTool
    PausePlayback      @6      ;necessary for SoundTool
    ContinuePlayback   @7      ;necessary for SoundTool
    PlayDlgProc        @8
  
```

PlaySetup routine gets called by SoundTool and asks user for parameters.

```

BOOL FAR PASCAL PlaySetup( HWND hWnd )
{
    FARPROC lpProcDialog;
    BOOL bReturn;

    lpProcDialog = MakeProcInstance( (FARPROC)PlayDlgProc, hInst);
    bReturn = DialogBox( hInst, "PLAY_DLG", hWnd, lpProcDialog);
    FreeProcInstance( lpProcDialog );

    return( bReturn );
}
  
```

```

void FAR PASCAL PlaySample( HWND hWnd, SAMPLE FAR * lpSample )
/*
 * hWnd is the window handle of SoundTool.
 * must post a WM_PLAYDONE message to SoundTool when done.
 * This allows SoundTool to repeat the sample if necessary.
 */
{
    DWORD dwLength;
    BYTE FAR * lpBuffer;

    if( NULL != lpSample->hGSound )
    {
        dwLength = min( lpSample->dwBytes,
                        (lpSample->dwStop - lpSample->dwStart) );

        if( 0L != dwLength )
        {
            if( NULL !=
                (lpBuffer = (BYTE HUGE *)GlobalWire( lpSample->hGSound )) )
            {
                GlobalPageLock( HIWORD(lpBuffer) );

                PlaySound( lpBuffer + lpSample->dwStart,
                           dwLength, lpSample->usFreq, lpSample->usSampleSize,
                           lpSample->usVolume, lpSample->usShift );

                GlobalPageUnlock( HIWORD(lpBuffer) );
                GlobalUnWire( lpSample->hGSound );
            }
        }
        PostMessage( hWnd, WM_PLAYDONE, 0, 0L );
    }
}

```

PlayDlgProc routine gets called by PlaySetup and asks user for parameters, does nothing in this sample library.

```

BOOL FAR PASCAL PlayDlgProc( HWND hDlg, unsigned message, WORD wParam, LONG
                             lParam)
{
    switch( message )
    {
        case WM_COMMAND:
            if( ID_OK == wParam )
            {
                EndDialog( hDlg, TRUE );
            }
            break;

        case WM_INITDIALOG:
            return( FALSE );
            break;
    }
    return( FALSE );
}

```

```
}
```

LibMain routine gets called by LIBENTRY.ASM when the DLL is loaded.

```
BOOL FAR PASCAL LibMain( HANDLE hInstance, WORD wDataSegment,
                        WORD cbHeapSize, LPSTR lpszCmdLine )
{
    hInst = hInstance;

    if( cbHeapSize > 0 )
        UnlockData( 0 );

    return( TRUE );
}
```

GetPLAYDLLVersion routine gets called by SoundTool when the DLL is loaded.

```
LONG FAR PASCAL GetPLAYDLLVersion( LPSTR lpBuffer, int nMaxLength )
/* *must* return PLAY_MAGIC in its loword */
/* hiword should contain a version number multiplied by 100 (100 == 1.00) */
/* if this is a background player or 0x8000 to the version number */
/* LONG FAR PASCAL GetPLAYDLLVersion() must have ordinal @4 */
{
    LoadString( hInst, S_MESSAGE+4, lpBuffer, nMaxLength );

    return( MAKELONG( VERSION, PLAY_MAGIC ) );
}
```

StopRecord gets called by SoundTool when [ESC] gets pressed.

```
BOOL FAR PASCAL StopPlayback( HWND hWnd )
{
    /* does nothing, but tells SoundTool about action */
    PostMessage( hWnd, WM_PLAYDONE, 0, 0L );
    return( TRUE );
}
```

```
BOOL FAR PASCAL PausePlayback( HWND hWnd )
{
    /* does nothing */
    return( TRUE );
}
```

```
BOOL FAR PASCAL ContinuePlayback( HWND hWnd )
{
    /* does nothing */
    return( TRUE );
}
```

GlobalVariables and **#defines** used by PLAYDLL

```
HANDLE hInst;                /* library instance handle */
char szAppName[] = "SoundTool"; /* for access to SNDTOOL.INI */
char szIniName[] = "SNDTOOL.INI"; /* for access to SNDTOOL.INI */
#define VERSION 100          /* == 1.00 */
#define PLAY_MAGIC 0x5059    /* == 'PY' */
#define WM_PLAYDONE WM_USER+1 /* do not change */
```

Makefile used to create PLAYDLL

```
all: playdll.dll

playdll.obj: playdll.c playdll.h
    cl -c -Asnw -Gsw -Oas -Zpe -FPi -W3 playdll.c

libentry.obj: libentry.asm
    masm -Mx libentry,libentry;

playdll.res:  playdll.rc playdll.dlg playdll.h
             rc -r playdll.rc

playdll.dll: libentry.obj playdll.obj playdll.def playdll.res
    link playdll+libentry, playdll.dll, /NOD /NOE sdllcew+libw+dsound,
        playdll.def
    rc playdll.res playdll.dll
```

PLAYDLL.DLG used to create PLAYDLL.RC

```
PLAY_DLG DIALOG LOADONCALL MOVEABLE DISCARDABLE 9, 26, 186, 42
CAPTION "Player Parameters"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "Nothing to set up !", -1, "static", SS_RIGHT | WS_GROUP |
        WS_CHILD, 8, 22, 76, 10
    CONTROL "&Ok", ID_OK, "button", BS_DEFPUSHBUTTON | WS_TABSTOP | WS_CHILD,
        134, 6, 46, 14
END
```

Adding a DLL for recording sound samples

SoundTool contains a mechanism that makes it possible for a Windows-programmer to incorporate recording subroutines by writing a DLL which conforms to the following interface. The library is loaded the same way as show in Adding a DLL for *playing* sound samples

If a recorder library is found, it is loaded into memory and the menu of SoundTool offers an additional item Settings → Recorder... to call a setup procedure inside this DLL; if no library was installed the button Record is inoperative.

To be callable from SoundTool the DLL must have at least seven exported functions with ordinal numbers @1 to @7:

@1 WEP, the usual Windows Exit procedure required by every dynamic link library

@2 This routine is called when the menuitem Settings → Record... is selected. The routine must confirm to the following calling sequence:

```
BOOL FAR PASCAL RecordSetup( HWND );
```

The routine should ask the user for all the recording parameters needed, and save them in the library data segment. The library is released when the user quits SoundTool, so it is advisable to store needed Parameters in WIN.INI. These parameters can be loaded when LibMain is called at load time of the library.

@3 This routine is called when the button Record is pressed. The routine must confirm to the following calling sequence:

```
BOOL FAR PASCAL RecordSample( HWND, SAMPLE FAR * );
```

The routine should global-allocate memory for the sampled data, record the sample and fill the SAMPLE structure with the corresponding data. The pointer to this structure is set up by SoundTool and must not be changed or modified. Just set all elements of the structure according to the definition above (Clipboard transfer). When the sampling process is finished you have to post a WM_RECDDONE message to SoundTool (even if there has been an error).

@4 This routine must return a LONG which has a version number its LOWORD and a magic word in the HIWORD (see example below). It is essential that the routine returns the magic number, otherwise SoundTool will refuse to load the library. Besides this return value the function must fill the supplied buffer with an ANSI string which identifies the input device. If the player is performing background sampling (returning immediately to SoundTool when starting to record a sample) the version number word must be ORed with 0x8000. The routine must confirm to the following calling sequence:

```
LONG FAR PASCAL GetRECDLLVersion( LPSTR, int )
```

@5 This procedure gets called, when the user presses the [ESC]-key. It returns a BOOL of TRUE if the record was successfully cancelled. It is necessary to inform SoundTool about the end of recording by posting a message of WM_RECDDONE too.

```
BOOL FAR PASCAL StopRecord( HWND )
```

@6 Not yet used in version 2.6. Returns a BOOL of TRUE if the recording was successfully paused. SoundTool waits until the recording is cancelled or continued.

```
BOOL FAR PASCAL PauseRecord( HWND )
```

@7 Not yet used in version 2.6. This procedure returns a BOOL which should be TRUE if the continuation of recording was successful.

```
BOOL FAR PASCAL ContinueRecord( HWND )
```

The following examples show excerpts from my sample RECDLL which uses an 8-bit A/D converter to sample audio data at up to 50 kHz. The library **must** contain at least the seven exported ordinals @1 to @7.

RECDLL.DEF file showing EXPORTS with ordinal numbers.

```
LIBRARY    RECDLL
EXETYPE    WINDOWS
CODE       PRELOAD MOVEABLE DISCARDABLE
DATA       MOVEABLE SINGLE
HEAPSIZE   1024
EXPORTS
    WEP                @1 RESIDENTNAME ;necessary for Windows
    RecordSetup        @2          ;necessary for SoundTool
    RecordSample       @3          ;necessary for SoundTool
    GetRECDLLVersion  @4          ;necessary for SoundTool
    StopRecord         @5          ;necessary for SoundTool
    PauseRecord        @6          ;necessary for SoundTool
    ContinueRecord     @7          ;necessary for SoundTool
    RecordDlgProc      @8          ;used internally by RECDLL
```

RecordSetup routine gets called by SoundTool and asks user for parameters.

```
BOOL FAR PASCAL RecordSetup( HWND hWnd )
{
    FARPROC lpProcDialog;
    BOOL bReturn;

    lpProcDialog = MakeProcInstance( (FARPROC)RecordDlgProc, hInst);
    bReturn = DialogBox( hInst, "RECORD_DLG", hWnd, lpProcDialog);
    FreeProcInstance( lpProcDialog );

    return( bReturn );    /* return TRUE if OK */
}
```

RecordSample routine gets called by SoundTool and returns the recorded data.

```
void FAR PASCAL RecordSample( HWND hWnd, SAMPLE FAR * lpSample )
{
    GLOBALHANDLE hGSound;
    BYTE HUGE * lpSound;
    BYTE HUGE * lpSoundStart;
    DWORD dwElapsed, dwStartTick, dwStopTick, dwBytes, dwFrequency;
    char szFormat[80];

    dwBytes = dwRecordBytes;

    if( NULL != (hGSound = GlobalAlloc( GMEM_MOVEABLE, dwBytes )) )
    {
        if( NULL != (lpSound = (BYTE HUGE *)GlobalWire( hGSound )) )
```

```

{
lpSample->hGSound = hGSound;
lpSample->dwBytes = dwBytes;
lpSample->dwStart = 0;
lpSample->dwStop = dwBytes;
lpSample->usSampleSize = 0;
lpSample->usVolume = 20;
lpSample->usShift = 4;
MessageBeep(0);
lpSoundStart = lpSound;
dwStartTick = GetTickCount(); /* ms */
if( nRecordDelay )
{
_asm
{
mov dx, usStartPort /* start first conversion */
out dx, al
}
while( dwBytes-- )
{
_asm
{
mov dx, usReadPort
in al, dx /* get a byte from A/D
converter */
les bx, DWORD PTR lpSound
mov BYTE PTR es:[bx], al
mov dx, usStartPort /* start next conversion */
out dx, al
mov cx, nRecordDelay /* loop counter */
WaitLoop:
loop WaitLoop /* empty loop */
}
lpSound++; /* increment huge pointer */
}
}
else /* fastest sampling rate */
{
_asm
{
mov dx, usStartPort /* start next conversion */
out dx, al
}
while( dwBytes-- )
{
/* get a byte from A/D converter */
_asm
{
mov dx, usReadPort
in al, dx /* get a byte from A/D */
les bx, DWORD PTR lpSound
mov BYTE PTR es:[bx], al
mov dx, usStartPort
out dx, al /* start conversion */
}
lpSound++; /* increment pointer */
}
}
}

```

```

    }
    dwStopTick = GetTickCount(); /* ms */
    MessageBeep(0);

    lpSound = lpSoundStart;
    dwBytes = lpSample->dwBytes;
    while( dwBytes-- )
    {
        if( *lpSound > 127 )
            *lpSound -= 128;
        else
            *lpSound += 128;
        *lpSound++;
    }
    GlobalUnWire( hGSound );

    if( dwStopTick < dwStartTick )
        dwElapsed = 0xffffffff - dwStartTick + dwStopTick;
    else
        dwElapsed = dwStopTick - dwStartTick;
    dwFrequency = (DWORD)(1000.0 * ((double)lpSample->dwBytes /
        (double)dwElapsed));
    lpSample->usFreq = (unsigned int)dwFrequency;
    LoadString( hInst, S_MESSAGE+3, szFormat, sizeof(szFormat)-1 );
    wsprintf( szBuffer, szFormat, dwFrequency );
    lstrcpy( lpSample->szName, szBuffer );
}
else
{
    /* cannot lock new sound bytes */
    GlobalFree( hGSound );
    lpSample->hGSound = NULL;
    ShowMessageBox( hWnd, 1, MB_OK );
}
}
else
{
    /* cannot allocate memory for new sound bytes */
    ShowMessageBox( hWnd, 2, MB_OK );
    lpSample->hGSound = NULL;
}
PostMessage( hWnd, WM_RECDONE, 0, 0L );
}

```

RecordDlgProc routine gets called by RecordSetup and asks user for parameters.

```

BOOL FAR PASCAL RecordDlgProc( HWND hDlg, unsigned message, WORD wParam, LONG
                                lParam)
{
    int nDelay;
    DWORD dwBytes;

    switch( message )
    {
        case WM_COMMAND:
            if( ID_OK == wParam )

```

```

    {
        GetDlgItemText( hDlg, ID_RECORDNUMBER, szBuffer, sizeof(szBuffer) );
    };
    dwBytes = (DWORD)atol( szBuffer );
    GetDlgItemText( hDlg, ID_RECORDDELAY, szBuffer,
        sizeof(szBuffer) );
    nDelay = atoi( szBuffer );

    if( nRecordDelay != nDelay )
    {
        nRecordDelay = nDelay;
        wsprintf( szBuffer, "%d", nRecordDelay );
        WritePrivateProfileString( szAppName, szRecordDelay, szBuffer,
            szIniFile );
    }

    if( dwRecordBytes != dwBytes )
    {
        dwRecordBytes = dwBytes;
        wsprintf( szBuffer, "%lu", dwRecordBytes );
        WritePrivateProfileString( szAppName, szRecordBytes, szBuffer,
            szIniFile );
    }
    EndDialog( hDlg, TRUE );
}
else if( ID_CANCEL == wParam )
{
    EndDialog( hDlg, FALSE );
}
break;

case WM_INITDIALOG:
    wsprintf( szBuffer, "%lu", dwRecordBytes );
    SetDlgItemText( hDlg, ID_RECORDNUMBER, szBuffer );
    wsprintf( szBuffer, "%d", nRecordDelay );
    SetDlgItemText( hDlg, ID_RECORDDELAY, szBuffer );
    SetFocus( GetDlgItem( hDlg, ID_RECORDDELAY ) );
    return( FALSE );
    break;
}
return( FALSE );
}

```

LibMain routine gets called by LIBENTRY.ASM when the DLL is loaded.

```

BOOL FAR PASCAL LibMain( HANDLE hInstance, WORD wDataSegment,
    WORD cbHeapSize, LPSTR lpszCmdLine )
{
    hInst = hInstance;

    /* get stored parameters from SNDTOOL.INI */
    GetPrivateProfileString( szAppName, szRecordBytes, "?",
        szBuffer, sizeof(szBuffer), szIniFile );
    if( '?' == szBuffer[0] )
    {
        /* no entry found, use default */
    }
}

```

```

        dwRecordBytes = 50000;
        wsprintf( szBuffer, "%lu", dwRecordBytes );
        WritePrivateProfileString( szAppName, szRecordBytes, szBuffer, szIniFile
            );
    }
    else
    {
        dwRecordBytes = (DWORD)atol( szBuffer );
    }
    GetPrivateProfileString( szAppName, szRecordDelay, "?",
        szBuffer, sizeof(szBuffer), szIniFile );
    if( '?' == szBuffer[0] )
    {
        /* no entry found, use default */
        nRecordDelay = 0;
        wsprintf( szBuffer, "%d", nRecordDelay );
        WritePrivateProfileString( szAppName, szRecordDelay, szBuffer, szIniFile
            );
    }
    else
    {
        nRecordDelay = atoi( szBuffer );
    }
    GetPrivateProfileString( szAppName, szRecordPort, "?",
        szBuffer, sizeof(szBuffer), szIniFile );
    if( '?' == szBuffer[0] )
    {
        /* no entry found, use default */
        usPort = 0x300;
        wsprintf( szBuffer, "%u", usPort );
        WritePrivateProfileString( szAppName, szRecordPort, szBuffer,
            szIniFile );
    }
    else
    {
        usPort = (unsigned int)atoi( szBuffer );
    }

    if( cbHeapSize > 0 )
        UnlockData( 0 );      /* make segment moveable */

    return( TRUE );
}

```

GetRECDLLVersion routine gets called by SoundTool when the DLL is loaded.

```

LONG FAR PASCAL GetRECDLLVersion( LPSTR lpBuffer, int nMaxLength )
{
    LoadString( hInst, S_MESSAGE+4, lpBuffer, nMaxLength );
    return( MAKELONG( VERSION, REC_MAGIC ) );
}

```

StopRecord gets called by SoundTool if the user presses [ESC].

```

BOOL FAR PASCAL StopRecord( HWND hWnd )
{
    /* does nothing */
    PostMessage( hWnd, WM_RECDONE, 0, 0L );
    return( TRUE );
}

```

```

BOOL FAR PASCAL PauseRecord( HWND hWnd )
{
    /* does nothing */
    return( TRUE );
}

```

```

BOOL FAR PASCAL ContinueRecord( HWND hWnd )
{
    /* does nothing */
    return( TRUE );
}

```

GlobalVariables and #defines used by RECDLL

```

#define VERSION 100                /* == 1.00 */
#define REC_MAGIC 0x5243           /* == 'RC' */
#define WM_RECDONE WM_USER+2
DWORD dwRecordBytes;              /* number of bytes to record */
int nRecordDelay;                 /* delay loop count */
HANDLE hInst;                     /* library instance handle */
unsigned int usPort;              /* A/D converter port address */
char szBuffer[80];                /* avoid buffer on stack DS != SS */
char szAppName[] = "SoundTool";
char szIniName[] = "SNDTOOL.INI"; /* for access to SNDTOOL.INI */
char szRecordBytes[] = "RecordBytes";
char szRecordDelay[] = "RecordDelay";
char szRecordPort[] = "RecordPort";

```

Makefile used to create RECDLL

```

all: recdll.dll

recdll.obj: recdll.c recdll.h
    cl -c -Asnw -Gsw -Oas -Zpe -FPi -W3 recdll.c

libentry.obj: libentry.asm
    masm -Mx libentry,libentry;

recdll.res:  recdll.rc recdll.dlg recdll.h
             rc -r recdll.rc

recdll.dll: libentry.obj recdll.obj recdll.def recdll.res
    link recdll+libentry, recdll.dll, /NOD /NOE sdllcew+libw, recdll.def
    rc recdll.res recdll.dll

```

RECDLL.DLG used to create RECDLL.RC

```
RECORD_DLG DIALOG LOADONCALL MOVEABLE DISCARDABLE 9, 26, 186, 42
CAPTION "Recording Parameters"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "&Number of samples:", -1, "static", SS_RIGHT | WS_GROUP |
        WS_CHILD, 10, 8, 74, 10
    CONTROL "", ID_RECORDNUMBER, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP |
        WS_CHILD, 90, 8, 32, 12
    CONTROL "&Delay count:", -1, "static", SS_RIGHT | WS_GROUP | WS_CHILD, 8,
        22, 76, 10
    CONTROL "", ID_RECORDDELAY, "edit", ES_LEFT | WS_BORDER | WS_TABSTOP |
        WS_CHILD, 90, 22, 32, 12
    CONTROL "&Cancel", ID_CANCEL, "button", BS_PUSHBUTTON | WS_GROUP |
        WS_TABSTOP | WS_CHILD, 134, 6, 46, 14
    CONTROL "&Ok", ID_OK, "button", BS_DEFPUSHBUTTON | WS_TABSTOP | WS_CHILD,
        134, 24, 46, 14
END
```

Using Dynamic Data Exchange (DDE)

SoundTool has a simple DDE interface which can be used to add audio capabilities to other applications which support DDE too.

You can execute commands within SoundTool or ask about its current state via DDE.

There is one Topic (General) and one item (State) which can be used in DDE communications. SoundTool is case insensitive, you can use General as well as GENERAL or gEnErAl.

Execute commands in SoundTool:

The user can record and playback samples by calling SoundTool via DDE.

There are two commands which can be sent as part of a *DDE-Execute* call:

```
Record.Data("soundname")  
Play.Data("soundname")
```

Each of the two calls needs one argument, a string which contains the name under which the sample is stored on disk (max. 8 characters plus path if necessary, no extension).

Because of disk space considerations DDE can be used with *packed sounds in *.SNP format* only !

When recording, SoundTool creates the packed file with an extension SNP and writes the sampled data to disk after the Record.Data command has been executed.

When playing a sample soundtool appends .SNP to "soundname" and tries to play this file.

Command strings must be enclosed in square brackets [].

Inquire information from SoundTool:

Current state:

The user can inquire the current state of SoundTool by sending a DDE-Request for the Item State. SoundTool will return a null terminated string containing one of the following three messages:

```
"idling",  
"recording" and  
"playing".
```

Version number:

The user can inquire the version number of SoundTool by sending a DDE-Request for the Item Version. SoundTool will return a null terminated string containing the Version number which, this Version returns:

```
"3.0"
```

Examples:

To communicate with SoundTool you have to perform the following steps:

1. open a DDE channel using the topic General.
2. send the appropriate command(s) once or multiple times
3. close the DDE channel

Example 1:

using a Microsoft Excel macro to record sound using an execute call:

```
1 INITIATE("SoundTool";"General") A B begin the DDE conversation
```

- | | |
|---|---|
| 2 | EXECUTE(A1;"[Record.Data("c:\tmp\msg1")]")execute the command |
| 3 | TERMINATE(A1) finish the conversation |

Example 2:

using a Microsoft Excel macro to get the current state of SoundTool:

- | A | B |
|---|--|
| 1 | INITIATE("SoundTool";"General") begin the DDE conversation |
| 2 | REQUEST(A1;"State") request the State info |
| 3 | TERMINATE(A1) finish the conversation |

It is possible to use a hot (or warm, depends on the application) link between an application and SoundTool instead of a macro. This is supported for the items which can be requested, in SoundTool version 2.2 there are only two: State and Version, where the last item is no hot candidate for a DDE-link because it will seldom change while your application is running.

Example 3:

using Microsoft Excel, putting the current state of SoundTool in the cell of a spreadsheet. The contents of the cell is automatically updated whenever the state of SoundTool changes:

- | A | B |
|---|---|
| 1 | = 'SoundTool' 'General' 'State' a hot link to SoundTool via DDE |

Example 4:

using a Microsoft WinWord macro to play sound using an execute call:

You can add sound annotations to your Documents by inserting a macrobutton field and an immediately following set bookmark field which sets a bookmark called Sound to the complete filename of the sound sample (which must be stored in packed format).

This looks like the following line if you have enabled field codes view:

```
{macrobutton PlaySample [Listen what Mama says]} {set Sound c:\\tmp\\w4w1}
```

You can have more than one of these field pairs, just insert them where you want. Of course you must have installed the following macro and loaded SoundTool (iconic preferred) before trying this.

Sub MAIN

REM Play a Sample, very basic example, lacks error handling

REM shows how to play a sample using WordBASIC

REM

REM skip to next field and select it (this should be the set bookmark field)

NextField

WordRight 1, 1

REM updating this field sets the bookmark Sound to the given value

UpdateFields

REM restore cursor to previous position

PrevField

REM get the bookmark text

Name\$ = **GetBookmark\$**("Sound")

REM play the sound if we were succesful

If Len(Name\$) > 0 **Then**

PlaySample(Name\$)

End If

End Sub

Sub PlaySample(Name\$)

REM plays the named sample cy calling SoundTool via DDE
REM

nChannel = **DDEInitiate**("SoundTool", "General")

REM I want to know what SoundTool is doing

State\$ = **DDERequest\$**(nChannel, "State")

REM just to give me a professional looking status bar

Version\$ = **DDERequest\$**(nChannel, "Version")

REM play the sample if SoundTool is waiting for me

If State\$ = "idling" **Then**

REM let's do it !

DDEExecute(nChannel, "[Play.Data(" + Chr\$(34) + Name\$ + Chr\$(34) +
"]")

State\$ = **DDERequest\$**(nChannel, "State")

Print "SoundTool " + Version\$ + " is " + State\$ + ", please wait..."

REM we are smart and wait until SoundTool is through

While State\$ = "playing"

State\$ = **DDERequest\$**(nChannel, "State")

Wend

Print "SoundTool " + Version\$ + " is " + State\$

Else

REM Hmm. nobody wants to listen to me

Print "SoundTool " + Version\$ + " is currently " + State\$ + ", please try
later"

End If

DDETerminate(nChannel)

End Sub

martin@mecha2.verfahrenstechnik.uni-stuttgart.de (INTERNET)

aaron@jessica.stanford.edu (INTERNET)

