

Help Index for SoundTool 2.1



[Info about SoundTool](#)



[The SoundTool window](#)



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



[Getting started with SoundTool](#)



[The SoundTool Command Line](#)



[The SoundTool menu](#)

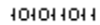


[Installing additional I/O libraries](#)



[Using Dynamic Data Exchange](#)

Informations for Programmers:



[File formats](#)

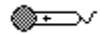


[Clipboard usage](#)

[Examples for clipboard data exchange](#)



[Adding a DLL for playing sounds](#)



[Adding a DLL for recording sounds](#)

SoundTool

Info about SoundTool 2.1




for Microsoft Windows Version 3

© 1990-1991 by Martin Hepperle

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

The included sound player relies on the dynamic link library DSOUND.DLL (© 1990-1991 by Aaron Wallace) to play a sound sample. To record samples you need either an AD-board or a Sound Blaster board (Sound Blaster support 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. Future enhancements could include a DDE interface to provide a basis for voice mail and including voice data into 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 \$15.- or DM 20.- to the following address to keep your nights peaceful and to receive the latest version of SoundTool which **includes additional libraries** which record and play using your **SoundBlaster** board.

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.

You may use the additional Sound Blaster library sndblst.dll on a single computer only, it is provided under license agreement of Creative Labs, which does not allow you to distribute this library.

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



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.

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.

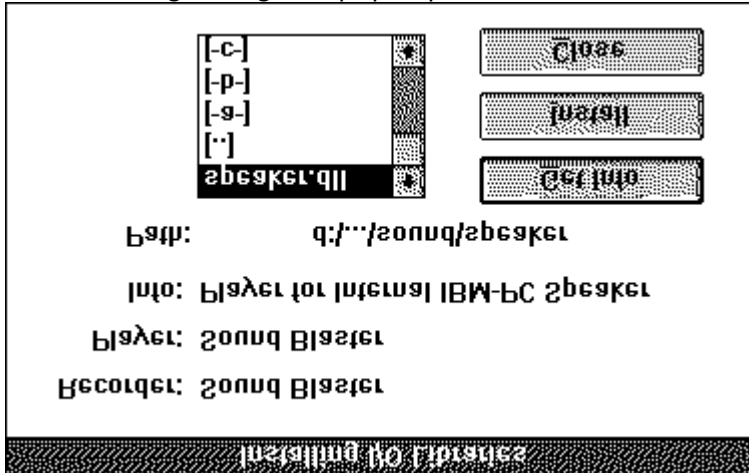
Some manufacturers like Creative Labs (maker of the widely used Sound Blaster board) are supplying drivers to use their hardware under Windows which makes it quite easy to write the additional interface library to interface with SoundTool's I/O functions.

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.

Getting started with SoundTool

If you did never use SoundTool before you have to select the libraries for voice input and output. 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.

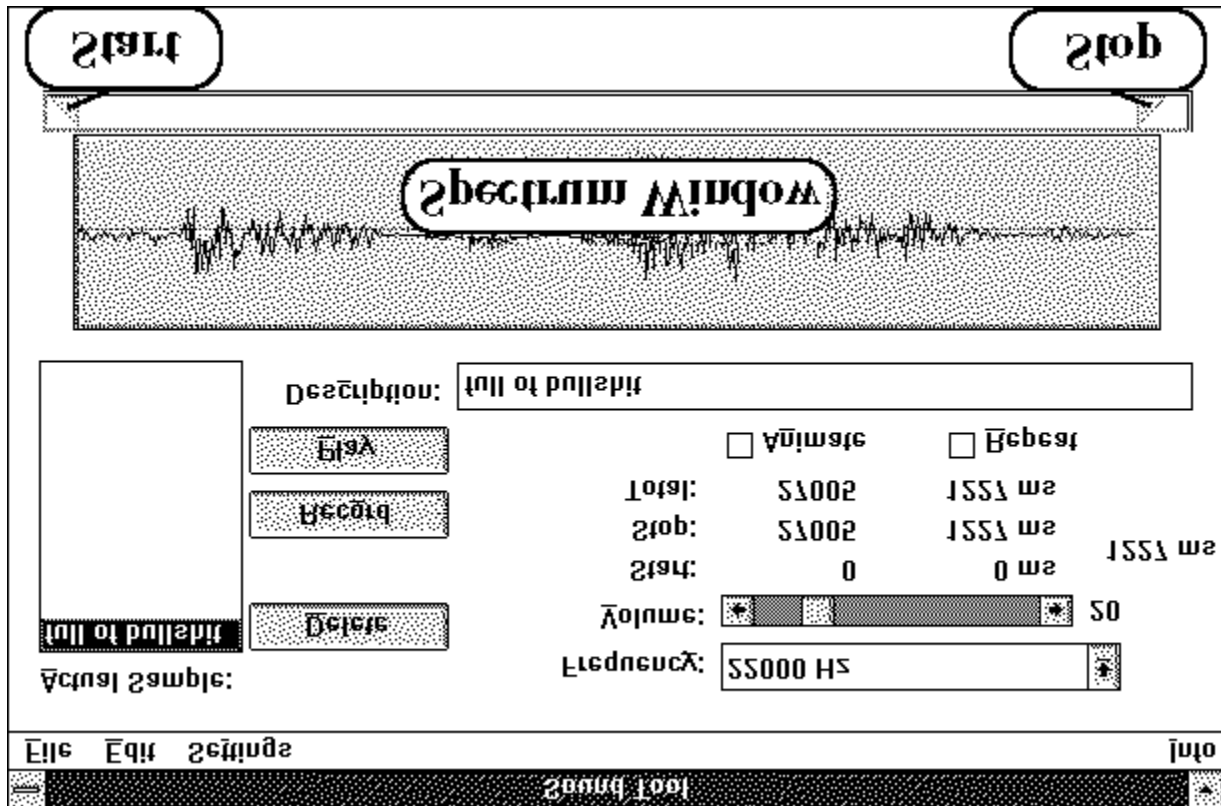
The SoundTool command line

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, SUN 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.
You can doubleclick onto a line containing a file with extension SOU, SND, SNP, TXT, SUN 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 !

The SoundTool window

After firing up SoundTool the following window is displayed:



Items in SoundTool window:

The window contains the following controls and displays:

- Actual Sample** a listbox which displays a list of loaded sounds. You can select one to be the 'actual sound' by clicking with the mouse on it.

- Delete** delete the actual 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 RECDLL.DLL installed). The sampling parameters can be set by the menu option 'Settings-Recorder...'


- Play** You can play the selected part of the actual sound by clicking on this button with the mouse. The player parameters can be set by the menu option 'Settings-Player...'

- 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. 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. 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 any key or by clicking the checkbox again while sound is playing. Do not press any other keys or try to switch to another application while the sound is playing, you can find yourself caught in an endless loop.
- Frequency** Select the frequency which is used to play the marked region of the sound by dragging the slider in the scrollbar or by clicking on the arrows at the end of the slider.
- Volume** Change the sound volume by dragging the elevator of this slider.
- 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 it's 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. 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 SoundTool's file formats (SND or SNP).
- Spectrum window** This display shows the spectrum of actual 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. Select the menuitem as usual by clicking the left mouse button or by moving the direction keys. The resulting display is shown below:

	ᾠλετσαλ	..0	
	ᾠβρευα	..A	
	ἰαετ	ἰαε	
ᾠβροσια	Ἰεω	Ἰουαυ	Ἰητ+ἰαε
Ἰελεσιου			
Ἰεβρεα: σοβλ			



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:

- ⇒ **8-Bit raw** *.SOU 8-Bit 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 automagically detects which kind of SND file it is reading.

⇒ Packed Sound

*.SNP 8-Bit sampled sound bytes compacted by a difference encoder.

⇒ NeXT

*.NXT one of the sound formats used by the NeXT computer. These files must contain raw 16-Bit samples which are reduced internally by SoundTool to 8-Bit. The NeXT μ Law-format is not supported yet.

⇒ SUN Audio

*.SUN sound format used by the SUN SparcStation. These files contain 8-Bit samples in μ Law-format which are reduced by SoundTool to 8-Bit.

⇒ 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.

⇒ File Save...

Writes the raw sound spectrum of the actual sound to a file. The various formats are explained above (File Open...).

⇒ File Install...

Used to install additional Dynamic Link Libraries for sound I/O boards. These Libraries have filenames with an extension .SLL.

Edit

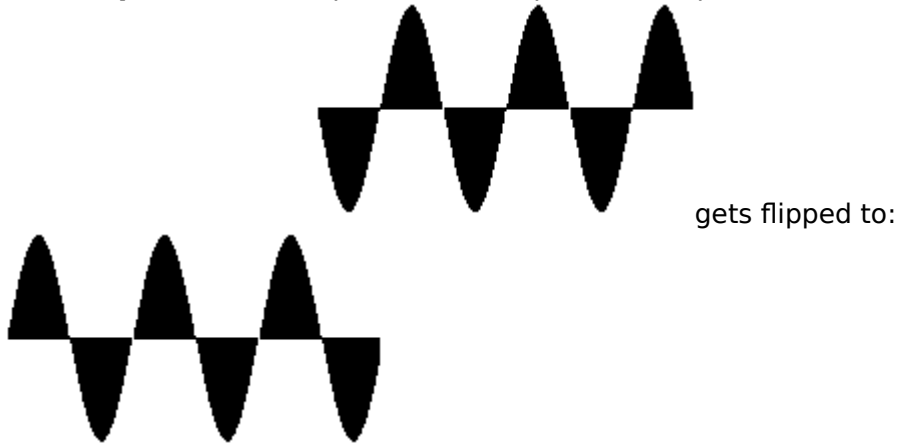
(This menu is available to quick access in the spectrum window, see: [the SoundTool window](#))

⇒ Repeat last command

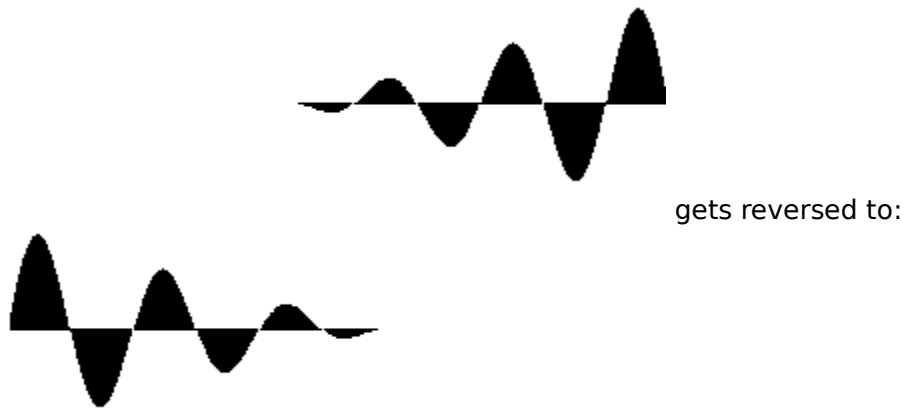
This option repeats the last command which is often easier then the access through the popup menu tree

⇒ **Edit Selection**
⇒ **Flip**

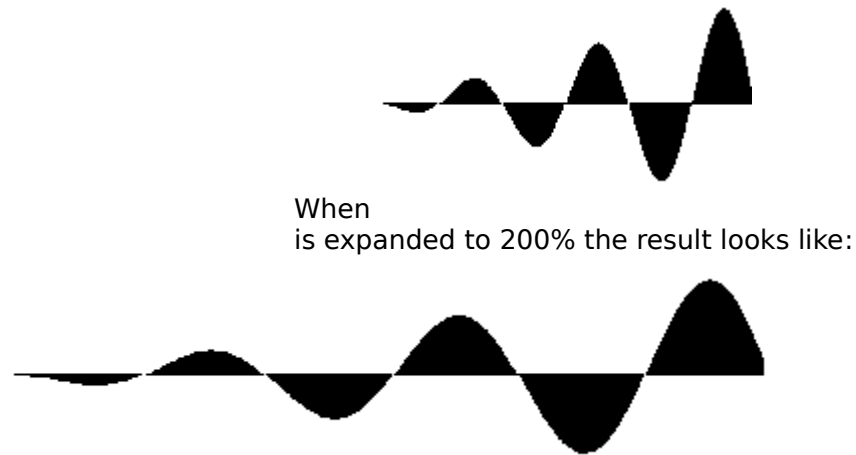
flips the marked part of the spectrum in vertical direction



⇒ **Reverse** reverses the byte sequence of the marked part of the spectrum



⇒ **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 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...'



is 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.



is faded in to:



- ⇒ **Fade Out** fades the bytes in the selected area of the spectrum by an 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 marked 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

⇒ **Edit Clipboard**

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

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.
- ⇒ **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%'.
- ⇒ **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 DLL installed).
- ⇒ **Player...** enter additional parameters if your sound board needs them. (Only applicable if you have a D/A-board and the matching player DLL 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.
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 more informations in it's header:

```
char szMagic[6] = { 'S', 'O', 'U', 'N', 'D', 0x1a }
GLOBALHANDLE hGSound;      /* not used */
DWORD dwBytes;              /* length of complete sample */
DWORD dwStart;              /* first byte to play from sample */
DWORD dwStop;              /* first byte NOT to play from sample */
WORD wFreq;                 /* frequency */
WORD wSampleSize;
WORD wVolume;
WORD wShift;
char szName[96];            /* name of sound */
```

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;      /* not used */
DWORD dwBytes;              /* length of complete sample */
DWORD dwStart;              /* first byte to play from sample */
DWORD dwStop;              /* first byte NOT to play from sample */
WORD wFreq;                 /* frequency */
WORD wSampleSize;
WORD wVolume;
WORD wShift;
char szName[96];            /* name of sound */
```

The compression method simply uses the difference between two succeeding bytes which is often below ± 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 the triple time for writing and the same time for reading, LHarc produced a file of 59 % and took the same threefold time to process the data).

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 actual 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 have one of the following values:
{ 5500, 7330, 11000, 22000 }

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. If an entry is found, it 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 inoperative.

To install a player library use the 'File → Install...' menu command.

A player DLL must have at least four exported functions with ordinal numbers @1, @2, @3 and @4 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 'WIN.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 it **must** post a message WM_PLAYDONE SoundTool's window; if you forget you will never leave SoundTool.

- @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 )
```

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    SPEAKERDLL
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
    PlaySample         @3
    GetPLAYDLLVersion @4
    PlayDlgProc        @5
```

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 )
        return;

    dwLength = min( lpSample->dwBytes,
                    (lpSample->dwStop - lpSample->dwStart) );

    if( 0L == dwLength )
        return;
```

```

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 (1000 == 10.00) */
/* 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) );
}

```

GlobalVariables and #defines used by PLAYDLL

```

HANDLE hInst;          /* library instance handle */
char szAppName[] = "SoundTool";
#define VERSION 100    /* == 1.00 */
#define PLAY_MAGIC 0x5059 /* == 'PY' */
#define WM_PLAYDONE WM_USER+1

```

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 four exported functions with ordinal numbers @1, @2, @3 and @4:

@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_RECDONE message tu SoundTool.

@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 )
```

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 four exported ordinals @1, @2, @3 and @4.

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
    RecordDlgProc @5 ;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 actual 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
            converter */
            les bx,DWORD PTR lpSound
            mov BYTE PTR es:[bx],al
            mov dx, usStartPort
            out dx, al
            mov cx, nRecordDelay
            WaitLoop:
            loop WaitLoop
        }
        lpSound++;
    }
}
else /* fastest sampling rate */
{
    _asm
    {
        mov dx, usStartPort
        out dx, al
    }
    while( dwBytes-- )
    {
        /* get a byte from A/D converter */
        _asm
        {
            mov dx, usReadPort
            in al, dx
            converter */
            les bx,DWORD PTR lpSound
            mov BYTE PTR es:[bx],al
            mov dx, usStartPort
            out dx, al
        }
        lpSound++;
    }
}
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 );
                    WriteProfileString( szAppName, szRecordDelay, szBuffer );
                }

                if( dwRecordBytes != dwBytes )

```

```

        {
            dwRecordBytes = dwBytes;
            wsprintf( szBuffer, "%lu", dwRecordBytes );
            WriteProfileString( szAppName, szRecordBytes, szBuffer );
        }
        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 WIN.INI */
    GetProfileString( szAppName, szRecordBytes, "?",
                    szBuffer, sizeof(szBuffer) );
    if( '?' == szBuffer[0] )
    {
        /* no entry found, use default */
        dwRecordBytes = 50000;
        wsprintf( szBuffer, "%lu", dwRecordBytes );
        WriteProfileString( szAppName, szRecordBytes, szBuffer );
    }
    else
    {
        dwRecordBytes = (DWORD)atol( szBuffer );
    }
    GetProfileString( szAppName, szRecordDelay, "?",
                    szBuffer, sizeof(szBuffer) );
    if( '?' == szBuffer[0] )
    {
        /* no entry found, use default */
        nRecordDelay = 0;
        wsprintf( szBuffer, "%d", nRecordDelay );
        WriteProfileString( szAppName, szRecordDelay, szBuffer );
    }
    else

```

```

    {
        nRecordDelay = atoi( szBuffer );
    }
    GetProfileString( szAppName, szRecordPort, "?",
                    szBuffer, sizeof(szBuffer) );
    if( '?' == szBuffer[0] )
    {
        /* no entry found, use default */
        usPort = 0x300;
        wsprintf( szBuffer, "%u", usPort );
        WriteProfileString( szAppName, szRecordPort, szBuffer );
    }
    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 ) );
}

```

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 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). SoundTool creates the file with an extension SNP and writes the sampled data to disk after the 'Record.Data' command has been executed.

Command strings must be enclosed in square brackets [].

Inquire information from SoundTool:

Actual 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:

```
"2.1"
```

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:

	A	B
1	INITIATE("SoundTool";"General")	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 2.1 there are only two: 'State' and 'Version'.

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:

- | | | | |
|---|---------------------------------|---|---------------------------------|
| 1 | A | 1 | B |
| | = '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 this WinBASIC
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

```


pacp@ds0rus1i.bitnet (This is a BITNET address)

or:

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

aaron@jessica.stanford.edu

