Help Index for SoundTool

Info about SoundTool

The SoundTool window

The SoundTool menu

Informations for Programmers:

- File formats
- Clipboard usage
- Examples for clipboard data exchange
- Adding a DLL for recording sounds

Info about SoundTool

for

Microsoft Windows

Version 3

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

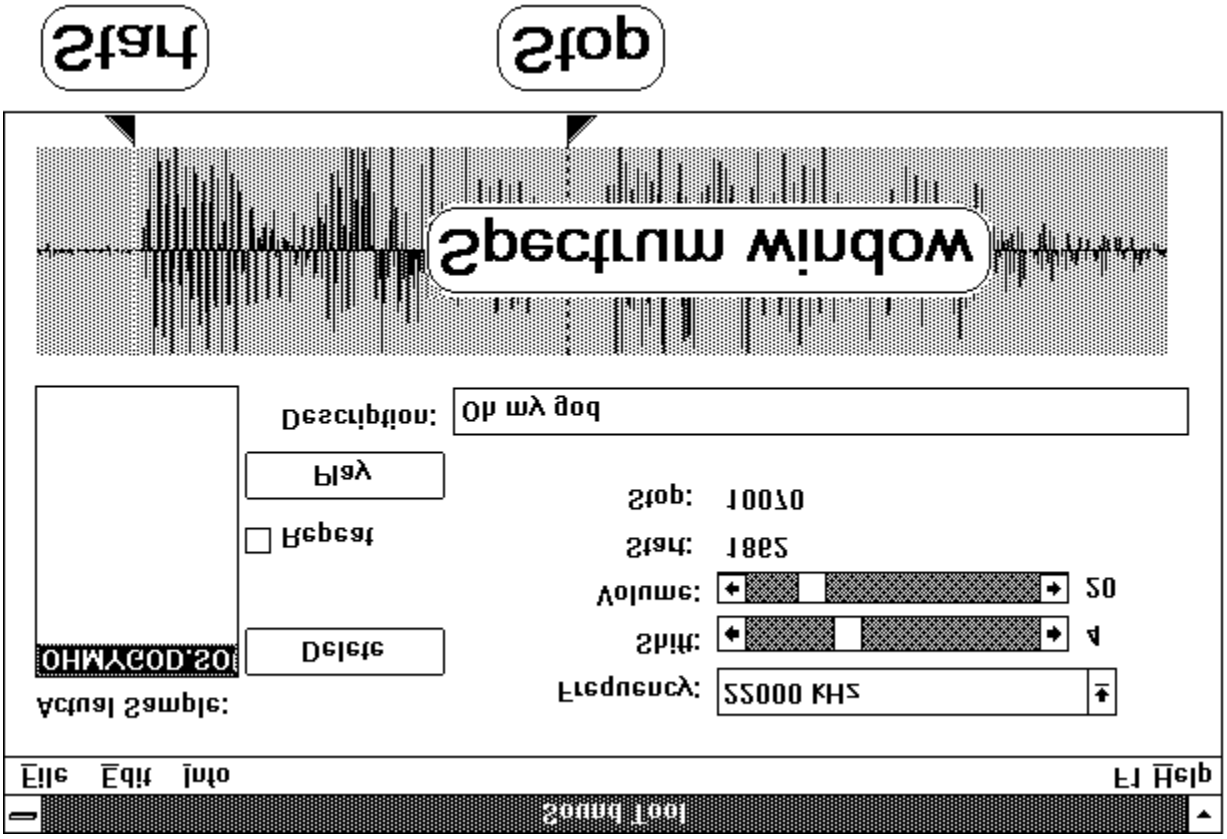
It relies on the dynamic link library DSOUND.DLL (© 1990-1991 by Aaron Wallace) to play a user selected part of the complete sound sample.

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.

To use SoundTool efficiently a  is necessary. Some functions cannot be used without such a beast.

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 sound — 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!

Play — You can play the selected part of the actual sound by clicking with the mouse on this button.

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 windo, 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.
Shift	Select a frequency shift with this scrollbar.
Volume	Change the sound volume by dragging the elevator of this slider.
Start	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 sprectum 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.
Stop	This is an indicator for the last selected byte; you can move it using the mouse in the same way as described above.
Description	This text string describes the sound by up to 95 characters.
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. The left slider indicates the start of a selection, the right one marks the end of the selection.

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 seleted 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.

Implementation follows when standard header format is defined:

- ⇒ **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.
- ⇒ **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 μLaw-format is not supported yet.
- ⇒ **SUN Audio** *.SUN sound format used by the SUN SparcStation. These files contain 8-Bit samples in μLaw-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
```

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

Edit

⇒ **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

gets flipped to:

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

gets reversed to:

⇒ **Expand** doubles each byte of the marked part of the spectrum, uses linear interpolation to produce a smooth result.
This has the same effect as a local decrease of frequency.

is expanded to:

⇒ **Shrink** collapses every two bytes of the marked part of the spectrum; uses average of the two bytes
This has the same effect as a local increase of frequency.

is shrinked to:

⇒ **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 ‘Preferences-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.

is faded out to:

⇒ **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 'Preferences-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.

Record Starts sampling of Sound data**.** (Only applicable if you have an A/D-board and matching RECDLL.DLL installed).

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.

⇒ **Record...** enter the number of samples to record and a loop delay count to adjust the recording frequency. (Only applicable if you have an A/D-card and matching RECDLL.DLL installed).

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

Info

⇒ **F1 Help** how the hell did you manage to display this help text ?

⇒ **Info...** same boring stuff as usual

File formats

SoundTool can save sound samples in various formats:

8 bit raw format — this is nothing more but a stream of contigout 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.

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  DESCR_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 clipbaord */
{
   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 clipbard, 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 finaly 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 recording sound samples

SoundTool contains a mechanism that makes it possible for a Windows-pogrammer to incorporate recording subroutines by writing a DLL which conforms to the following interface. Whenever SoundTool is started, it looks for a file 'RECDLL.DLL'. If a file of this name is found in the directory where SoundTool resides, it is loaded into memory and the menu of SoundTool offers two items to call into this DLL:

- Record
- Settings → Record...

To be callable from SoundTool the DLL must have at least two exported functions with ordinal numbers @3 and @2:

@3 This routine is called when the menuitem 'Record' is selected. 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).

@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.

The following examples shows excerpts from my sample RECDLL which uses an 8-bit A/D converter to sample audio data at up to 40 kHz. The library **must** be named RECDLL.DLL and **must** contain at least the two exported ordinals @2 and @3.

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
    RecordDlgProc       @4     ;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.

```
BOOL FAR PASCAL RecordSample( HWND hWnd, SAMPLE FAR * pSound )
{
   GLOBALHANDLE hGSound;
   BYTE HUGE * lpSound;
   BYTE HUGE * lpSoundStart;
   DWORD dwElapsed, dwStartTick, dwStopTick, dwBytes, dwFrequency;
   BOOL bReturn;

   bReturn = FALSE;

   dwBytes = dwRecordBytes;

   if( NULL != (hGSound = GlobalAlloc( GMEM_MOVEABLE, dwBytes )) )
     {
      if( NULL != (lpSound = (BYTE HUGE *)GlobalLock( hGSound )) )
        {
         pSound->hGSound = hGSound;
         pSound->dwBytes = dwBytes;
         pSound->dwStart = 0;
         pSound->dwStop  = dwBytes;
         pSound->usFreq  = 22000;
         pSound->usSampleSize = 0;
         pSound->usVolume = 20;
         pSound->usShift  = 4;
         MessageBeep(0);
         lpSoundStart = lpSound;
         dwStartTick = GetTickCount(); /* ms */
         if( nRecordDelay )
           {
            outp( (usPort+1), 0x0000 );      /* start conversion */
            while( dwBytes-- )
              {
               /* get a byte from A/D converter */
               *lpSound++ = (BYTE)inp( usPort );
               outp( (usPort+1), 0x0000 );      /* start next conversion */
               _asm
                 {
                  mov cx, nRecordDelay               /* loop counter */
                  WaitLoop:
                  loop WaitLoop                      /* empty loop */
                 }
              }
           }
```

```
         else
           {
            outp( (usPort+1), 0x0000 );      /* start conversion */
            while( dwBytes-- )
              {
               /* get a byte from A/D converter */
               *lpSound++ = (BYTE)inp( usPort );
               outp( (usPort+1), 0x0000 );      /* start next conversion */
              }
           }
         dwStopTick = GetTickCount(); /* ms */
         MessageBeep(0);

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

         /* elapsed time in milliseconds */
         if( dwStopTick < dwStartTick )
            dwElapsed = 0xffffffff - dwStartTick + dwStopTick;
         else
            dwElapsed = dwStopTick - dwStartTick;
         dwFrequency = (DWORD)(1000.0 * ((double)pSound->dwBytes /
                   (double)dwElapsed));
         pSound->usFreq  = (unsigned int)dwFrequency;
         wsprintf( szBuffer, "Recorded at %u [Hz]", dwFrequency );
         lstrcpy( pSound->szName, szBuffer );
         bReturn = TRUE;
        }
       else
         {
          /* cannot lock new sound bytes */
          GlobalFree( hGSound );
         }
     }
   else
     {
      /* cannot allocate memory for new sound bytes */
     }

   return( bReturn );
}
```

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 );
}
```

GlobalVariables used by RECDLL

```
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
```

---

pacp@ds0rus1i.bitnet

aaron@jessica.stanford.edu