Demonstration Program SoundAndSpeech Listing

```
// ****************************************************************************************
// SoundAndSpeech.c                                                        CLASSIC EVENT MODEL
// ****************************************************************************************
//
// This program opens a modeless dialog containing five bevel button controls arranged in
// two groups, namely, a synchronous sound group and an asynchronous sound group.  Clicking on
// the bevel buttons causes sound to be played back or recorded as follows:
//
// •  Synchronous group:
//
//    •  Play sound resource.
//
//    •  Record sound resource (Mac OS 8/9 only).
//
//    •  Speak text string.
//
// •  Asynchronous group:
//
//    •  Play sound resource.
//
//    •  Speak text string.
//
// The asynchronous sound sections of the program utilise a special library called
// AsyncSoundLibPPC, which must be included in the CodeWarrior project.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  A 'DLOG' resource and associated 'DITL', 'dlgx', and 'dftb' resources (all purgeable).
//
// •  'CNTL' resources (purgeable) for the controls within the dialog.
//
// •  Two 'snd ' resources, one for synchronous playback (purgeable) and one for asynchronous
//    playback (purgeable).
//
// •  Four 'cicn' resources (purgeable).  Two are used to provide an animated display which
//    halts during synchronous playback and continues during asynchronous playback.  The
//    remaining two are used by the bevel button controls.
//
// •  Two 'STR#' resources containing "speak text" strings and error message strings (all
//    purgeable).
//
// •  'hrct' and 'hwin' resources (purgeable) for balloon help.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// Each time it is invoked, the function doRecordResource creates a new 'snd' resource with a
// unique ID in the resource fork of a file titled "SoundResources".
//
// ****************************************************************************************

// ..... includes

#include <Carbon.h>
#include <string.h>

// ..... defines

#define rDialog                 128
#define  iDone                  1
#define  iPlayResourceSync      4
#define  iRecordResource        5
#define  iSpeakTextSync         6
#define  iPlayResourceASync     7
```

```
#define  iSpeakTextAsync        8
#define rPlaySoundResourceSync  8192
#define rPlaySoundResourceASync 8193
#define rSpeechStrings          128
#define rErrorStrings           129
#define  eOpenDialogFail        1
#define  eCannotInitialise      2
#define  eGetResource           3
#define  eMemory                4
#define  eMakeFSSpec            5
#define  eWriteResource         6
#define  eNoChannelsAvailable   7
#define  ePlaySound             8
#define  eSndPlay               9
#define  eSndRecord             10
#define  eSpeakString           11
#define rColourIcon1            128
#define rColourIcon2            129
#define kMaxChannels            8
#define kOutOfChannels          1

// ..................................................................................... global variables

Boolean   gRunningOnX = false;
Boolean   gDone;
DialogRef gDialogRef;

// ..................................................... AsyncSoundLib attention flag

Boolean  gCallAS_CloseChannel = false;

// ..................................................................................... function prototypes

void  main                (void);
void  doPreliminaries     (void);
OSErr quitAppEventHandler  (AppleEvent *,AppleEvent *,SInt32);
void  doInitialiseSoundLib (void);
void  eventLoop           (void);
void  doDialogHit         (SInt16);
void  doPlayResourceSync  (void);
void  doRecordResource    (void);
void  doSpeakStringSync   (void);
void  doPlayResourceASync  (void);
void  doSpeakStringAsync  (void);
void  doSetUpDialog       (void);
void  doErrorAlert        (SInt16);
void  helpTags            (DialogRef);

// ..................................................................... AsyncSoundLib function prototypes

OSErr  AS_Initialise   (Boolean *,SInt16);
OSErr  AS_GetChannel   (SInt32,SndChannelPtr *);
OSErr  AS_PlayID       (SInt16, SInt32 *);
OSErr  AS_PlayHandle   (Handle,SInt32 *);
void   AS_CloseChannel (void);
void   AS_CloseDown    (void);

// ********************************************************************************** main

void main(void)
{
  SInt32 response;

  // ..................................................................................... do preliminaries

  doPreliminaries();

  // ..................................................... disable Quit item in Mac OS X Application menu
```

```
  DisableMenuCommand(NULL,'quit');

  // ..... open and set up dialog

  if(!(gDialogRef = GetNewDialog(rDialog,NULL,(WindowRef) -1)))
  {
    doErrorAlert(eOpenDialogFail);
    ExitToShell();
  }

  SetPortDialogPort(gDialogRef);
  SetDialogDefaultItem(gDialogRef,kStdOkItemIndex);

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    helpTags(gDialogRef);
    gRunningOnX = true;
  }

  doSetUpDialog();

  ShowWindow(GetDialogWindow(gDialogRef));
  // ..... initialise AsyncSoundLib

  doInitialiseSoundLib();

  // ..... enter event loop

  eventLoop();
}

// ********************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(64);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                            NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                            0L,false);
  if(osError != noErr)
    ExitToShell();
}

// ********************************************************************** doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
```

```
}

// ********************************************************************** doInitialiseSoundLib

void  doInitialiseSoundLib(void)
{
  if(AS_Initialise(&gCallAS_CloseChannel,kMaxChannels) != noErr)
  {
    doErrorAlert(eCannotInitialise);
    ExitToShell();
  }
}

// ************************************************************************************ eventLoop

void  eventLoop(void)
{
  Rect        theRect, eraseRect;
  CIconHandle colourIconHdl1;
  CIconHandle colourIconHdl2;
  SInt16      fontNum, itemHit;
  UInt32      finalTicks;
  Boolean     gotEvent;
  EventRecord eventStructure;
  DialogRef   theDialogRef;
  WindowRef   windowRef;
  SInt16      partCode;

  SetRect(&theRect,262,169,294,201);
  SetRect(&eraseRect,310,170,481,200);
  colourIconHdl1 = GetCIcon(rColourIcon1);
  colourIconHdl2 = GetCIcon(rColourIcon2);

  gDone = false;

  while(!gDone)
  {
    if(gCallAS_CloseChannel)
    {
      AS_CloseChannel();

      GetFNum("\pGeneva",&fontNum);
      TextFont(fontNum);
      TextSize(10);
      MoveTo(341,189);
      DrawString("\pAS_CloseChannel called");
      QDFlushPortBuffer(GetWindowPort(FrontWindow()),NULL);
      Delay(45,&finalTicks);
    }

    gotEvent = WaitNextEvent(everyEvent,&eventStructure,10,NULL);

    if(gotEvent)
    {
      if(IsDialogEvent(&eventStructure))
      {
        if(DialogSelect(&eventStructure,&theDialogRef,&itemHit))
          doDialogHit(itemHit);
      }
      else
      {
        if(eventStructure.what == mouseDown)
        {
          partCode = FindWindow(eventStructure.where,&windowRef);
          if(partCode == inDrag)
            DragWindow(windowRef,eventStructure.where,NULL);
          if(partCode == inMenuBar)
            MenuSelect(eventStructure.where);
        }
```

```
      }
    }
    else
    {
      PlotCIcon(&theRect,colourIconHdl1);
      QDFlushPortBuffer(GetDialogPort(gDialogRef),NULL);
      Delay(5,&finalTicks);
      PlotCIcon(&theRect,colourIconHdl2);
      Delay(5,&finalTicks);
      EraseRect(&eraseRect);
    }
  }

  DisposeDialog(gDialogRef);

  AS_CloseDown();
}

// ********************************************************************************** doDialogHit

void  doDialogHit(SInt16 item)
{
  switch(item)
  {
    case iDone:
      gDone = true;
      break;

    case iPlayResourceSync:
      doPlayResourceSync();
      break;

    case iRecordResource:
      doRecordResource();
      break;

    case iSpeakTextSync:
      doSpeakStringSync();
      break;

    case iPlayResourceASync:
      doPlayResourceASync();
      break;

    case iSpeakTextAsync:
      doSpeakStringAsync();
      break;
  }
}

// ************************************************************************** doPlayResourceSync

void  doPlayResourceSync(void)
{
  SndListHandle sndListHdl;
  SInt16        resErr;
  OSErr         osErr;
  ControlRef    controlRef;

  sndListHdl = (SndListHandle) GetResource('snd ',rPlaySoundResourceSync);
  resErr = ResError();
  if(resErr != noErr)
    doErrorAlert(eGetResource);

  if(sndListHdl != NULL)
  {
    HLock((Handle) sndListHdl);
    osErr = SndPlay(NULL,sndListHdl,false);
    if(osErr != noErr)
```

```
      doErrorAlert(eSndPlay);
    HUnlock((Handle) sndListHdl);
    ReleaseResource((Handle) sndListHdl);

    GetDialogItemAsControl(gDialogRef,iPlayResourceSync,&controlRef);
    SetControlValue(controlRef,0);
  }
}

// ************************************************************************** doRecordResource

void  doRecordResource(void)
{
  SInt16     oldResFileRefNum, theResourceID, resErr, tempResFileRefNum;
  BitMap     screenBits;
  Point      topLeft;
  OSErr      memErr, osErr;
  Handle     soundHdl;
  FSSpec     fileSpecTemp;
  ControlRef controlRef;

  oldResFileRefNum = CurResFile();

  GetQDGlobalsScreenBits(&screenBits);
  topLeft.h = (screenBits.bounds.right / 2) - 156;
  topLeft.v = 150;

  soundHdl = NewHandle(25000);
  memErr = MemError();
  if(memErr != noErr)
  {
    doErrorAlert(eMemory);
    return;
  }

  osErr = FSMakeFSSpec(0,0,"\pSoundResources",&fileSpecTemp);
  if(osErr == noErr)
  {
    tempResFileRefNum = FSpOpenResFile(&fileSpecTemp,fsWrPerm);
    UseResFile(tempResFileRefNum);
  }
  else
    doErrorAlert(eMakeFSSpec);

  if(osErr == noErr)
  {
    osErr = SndRecord(NULL,topLeft,siBetterQuality,&(SndListHandle) soundHdl);
    if(osErr != noErr && osErr != userCanceledErr)
      doErrorAlert(eSndRecord);
    else if(osErr != userCanceledErr)
    {
      do
      {
        theResourceID = UniqueID('snd ');
      } while(theResourceID <= 8191 && theResourceID >= 0);

      AddResource(soundHdl,'snd ',theResourceID,"\pTest");
      resErr = ResError();
      if(resErr == noErr)
        UpdateResFile(tempResFileRefNum);
      resErr = ResError();
      if(resErr != noErr)
        doErrorAlert(eWriteResource);
    }

    CloseResFile(tempResFileRefNum);
  }

  DisposeHandle(soundHdl);
```

```
  UseResFile(oldResFileRefNum);

  GetDialogItemAsControl(gDialogRef,iRecordResource,&controlRef);
  SetControlValue(controlRef,0);
}

// ************************************************************************** doSpeakStringSync

void  doSpeakStringSync(void)
{
  SInt16     activeChannels;
  Str255     theString;
  OSErr      resErr, osErr;
  ControlRef controlRef;

  activeChannels = SpeechBusy();

  GetIndString(theString,rSpeechStrings,1);
  resErr = ResError();
  if(resErr != noErr)
  {
    doErrorAlert(eGetResource);
    return;
  }

  osErr = SpeakString(theString);
  if(osErr != noErr)
    doErrorAlert(eSpeakString);

  while(SpeechBusy() != activeChannels)
    ;

  GetDialogItemAsControl(gDialogRef,iSpeakTextSync,&controlRef);
  SetControlValue(controlRef,0);
}

// ************************************************************************** doPlayResourceASync

void  doPlayResourceASync(void)
{
  SInt16 error;

  error = AS_PlayID(rPlaySoundResourceASync,NULL);
  if(error == kOutOfChannels)
    doErrorAlert(eNoChannelsAvailable);
  else
    if(error != noErr)
      doErrorAlert(ePlaySound);
}

// ************************************************************************** doSpeakStringAsync

void  doSpeakStringAsync(void)
{
  Str255 theString;
  OSErr  resErr, osErr;

  GetIndString(theString,rSpeechStrings,2);
  resErr = ResError();
  if(resErr != noErr)
  {
    doErrorAlert(eGetResource);
    return;
  }

  osErr = SpeakString(theString);
  if(osErr != noErr)
    doErrorAlert(eSpeakString);
}
```

```
// ************************************************************************** doSetUpDialog

void  doSetUpDialog(void)
{
  SInt16                        a;
  Point                         offset;
  ControlRef                    controlRef;
  ControlButtonGraphicAlignment alignConstant = kControlBevelButtonAlignLeft;
  ControlButtonTextPlacement    placeConstant = kControlBevelButtonPlaceToRightOfGraphic;

  offset.v = 1;
  offset.h = 5;

  for(a=iPlayResourceSync;a<iSpeakTextAsync+1;a++)
  {
    GetDialogItemAsControl(gDialogRef,a,&controlRef);
    SetControlData(controlRef,kControlEntireControl,kControlBevelButtonGraphicAlignTag,
                   sizeof(alignConstant),&alignConstant);
    SetControlData(controlRef,kControlEntireControl,kControlBevelButtonGraphicOffsetTag,
                   sizeof(offset),&offset);
    SetControlData(controlRef,kControlEntireControl,kControlBevelButtonTextPlaceTag,
                  sizeof(placeConstant),&placeConstant);
  }
  if(gRunningOnX)
  {
    GetDialogItemAsControl(gDialogRef,iRecordResource,&controlRef);
    DeactivateControl(controlRef);
  }
}

// ************************************************************************** doErrorAlert

void  doErrorAlert(SInt16 errorStringIndex)
{
  Str255 errorString;
  SInt16 itemHit;

  GetIndString(errorString,rErrorStrings,errorStringIndex);

  StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
}

// ****************************************************************************** helpTags

void  helpTags(DialogRef dialogRef)
{
  HMHelpContentRec helpContent;
  SInt16           a;
  ControlRef       controlRef;

  memset(&helpContent,0,sizeof(helpContent));
  HMSetTagDelay(500);
  HMSetHelpTagsDisplayed(true);

  helpContent.version = kMacHelpVersion;
  helpContent.tagSide = kHMOutsideTopCenterAligned;
  helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
  helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 130;

  for(a = 1;a <= 5; a++)
  {
    if(a == 2)
      continue;
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = a;
    GetDialogItemAsControl(dialogRef,a + 3,&controlRef);
    HMSetControlHelpContent(controlRef,&helpContent);
  }
}
```

```
// ********************************************************************************************
```

Demonstration Program SoundAndSpeech Comments

On Mac OS 9, ensure that the Speech Manager extension is on before running this program.

When this program is run, the user should click on the various buttons in the dialog box to play back and record sound resources and to play back the provided "speak text" strings. The user should observe the effects of asynchronous and synchronous playback on the "working man" icon in the image well in the dialog. The user should also observe that the text "AS_CloseChannel called" appears briefly in the secondary group box to the right of the image well when AsynchSoundLib sets the application's "attention" flag to true, thus causing the application to call the AsynchSoundLib function AS_CloseChannel.

Note that the doRecordResource function saves recorded sounds as 'snd ' resources with unique IDs in the resource fork of the file titled "SoundResources".

defines

kMaxChannels will be used to specify the maximum number of sound channels that AsynchSoundLib is to open. kOutOfChannels will be used to determine whether the AsynchSoundLib function AS_PlayID returns a "no channels available" error.

main

doInitialiseSoundLib is called to initialise the AsynchSoundLib library.

doInitialiseSoundLib

doInitialiseSoundLib initialises the AsynchSoundLib library. More specifically, it calls the AsynchSoundLib function AS_Initialise and passes to AsynchSoundLib the address of the application's "attention" flag (gAS_CloseChannel), together with the requested number of channels.

If AS_Initialise returns a non-zero value, an error alert is displayed and the program terminates.

eventLoop

Within the event loop, the "attention" flag (gCallAS_CloseChannel) required by AsynchSoundLib is checked. If AsynchSoundLib has set it to true, the AsynchSoundLib function AS_CloseChannel is called to free up the relevant ASStructure, close the relevant sound channel, and clear the "attention" flag. In addition, some text is drawn in the group box to the right of the image well to indicate to the user that AS_CloseChannel has just been called.

If WaitNextEvent returns other than zero, IsDialogEvent is called to determine whether the event belongs to the dialog. If so, DialogSelect is called to determine whether one of the dialog's buttons was clicked. If so, the function doDialogHit is called to further process the item hit. If the event does not belong to the dialog, the else block supports dragging of the dialog and choosing Show/Hide Balloons from the Help menu.

If zero was returned by WaitNextEvent, the two frames of "working man" animation are drawn within the image well, separated by five ticks, and the area in which "AS_CloseChannel called" may have been drawn is erased.

When gDone is set to true, the event loop exits, the dialog is disposed of, and the AsynchSoundLib function AS_CloseDown is called to stop all current playback, close open sound channels, and dispose of the associated ASStructures.

doPlayResourceSync

doPlayResourceSync is the first of the synchronous playback functions. It uses SndPlay to play a specified 'snd ' resource.

GetResource attempts to load the resource. If the subsequent call to ResError indicates an error, an error alert is presented.

If the load was successful, the sound handle is locked prior to a call to SndPlay. Since NULL is passed in the first parameter of the SndPlay call, SndPlay automatically allocates a sound channel to play the sound and deallocates the channel when the playback is complete. false passed in the third parameter specifies that the playback is to be synchronous.

> Note: The 39940-byte 'snd ' resource being used contains one command only (bufferCmd). The compressed sound header indicates MACE 3:1 compression. The sound length is 119568 frames. The 8-bit mono sound was sampled at 22kHz.

SndPlay causes all commands and data contained in the sound handle to be sent to the channel. Since there is a bufferCmd command in the 'snd ' resource, the sound is played.

If SndPlay returns an error, an error alert is presented.

When SndPlay returns, HUnlock unlocks the sound handle and ReleaseResource releases the resource.

doRecordResource

doRecordResource uses SndRecord to record a sound synchronously and then saves the sound in a 'snd ' resource. The 'snd ' resource will be saved to the resource fork of the file "SoundResources".

The first line saves the file reference number of the current resource file. The next three lines establish the location for the top left corner of the sound recording dialog.

NewHandle creates a relocatable block. The address of the handle will be passed as the fourth parameter of the SndRecord call. The size of this block determines the recording time available. (If NULL is passed as the fourth parameter of a SndRecord call, the Sound Manager allocates the largest block possible in the application's heap.) If NewHandle cannot allocate the block, an error alert is presented and the function returns.

The next block opens the resource fork of the file "SoundResources" and makes it the current resource file.

SndRecord opens the sound recording dialog and handles all user interaction until the user clicks the Cancel or Save button. Note that the second parameter of the SndRecord call establishes the location for the top left corner of the sound recording dialog and that the third parameter specifies 22kHz, mono, 3:1 compression.

When the user clicks the Save button, the handle is resized automatically. If the user clicks the Cancel button, SndRecord returns userCanceledErr. If SndRecord returns an error other than userCanceledErr, an error alert is presented and the function returns after closing the resource fork of the file, disposing of the relocatable block, and restoring the saved resource file reference number.

The relocatable block allocated by NewHandle, and resized as appropriate by SndPlay, has the structure of a 'snd ' resource, but its handle is not a handle to an existing resource. To save the recorded sound as a 'snd ' resource in the resource fork of the current resource file, the do/while loop first finds an acceptable unique resource ID for the resource. (For the System file, resource IDs for 'snd ' resources in the range 0 to 8191 are reserved for use by Apple Computer, Inc. Avoiding those IDs in this demonstration is not strictly necessary, since there is no intention to move those resources to the System file.)

The call to AddResource causes the Resource Manager to regard the relocatable block containing the sound as a 'snd ' resource. If the call is successful, UpdateResFile writes the changed resource map and the 'snd ' resource to disk. If an error occurs, an error alert is presented.

The relocatable block is then disposed of, the resource fork of the file "SoundResources" is closed, and the saved resource file reference number is restored.

doSpeakStringSync

doSpeakStringSync uses SpeakString to speak a specified string resource and takes measures to cause the speech to be generated in a psuedo-synchronous manner.

The speech that SpeakString generates is asynchronous, that is, control returns to the application before SpeakString finishes speaking the string. In this function, SpeechBusy is used to cause the speech activity to be synchronous so far as the function as a whole is concerned. That is, doSpeakStringSync will not return until the speech activity is complete.

As a first step, the first line saves the number of speech channels that are active immediately before the call to SpeakString.

GetIndString loads the first string from the specified 'STR#' resource. If an error occurs, an error alert is presented and the function returns.

SpeakString, which automatically allocates a speech channel, is called to speak the string. If SpeakString returns an error, an error alert is presented.

Although SpeakString returns control to the application immediately it starts generating the speech, the speech channel it opens remains open until the speech concludes. While the speech continues, the number

of speech channels open will be one more that the number saved at the first line. Accordingly, the while loop continues until the number of open speech channels is equal to the number saved at the first line. Then, and only then, does doSpeakStringSync exit.

doPlayResourceASync

doPlayResourceASync uses the AsynchSoundLib function AS_PlayID to play a 'snd ' resource asynchronously.

> Note: The 24194-byte 'snd ' resource being used contains one command only (bufferCmd). The compressed sound header indicates no compression. The sound length is 24195 frames. The 8-bit mono sound was sampled at 5kHz.

AS_PlayID is called to play the 'snd ' resource specified in the first parameter. Since no further control over the playback is required, NULL is passed in the second parameter. (Recall that, if you pass a pointer to a variable in the second parameter, AS_PlayID returns a reference number in that parameter. That reference number may be used to gain more control over the playback process. If you simply want to trigger a sound and let it to run to completion, you pass NULL in the second parameter, in which case a reference number is not returned by AS_PlayID.)

If AS_PlayID returns the "no channels currently available" error, an error alert is presented advising of that specific condition. If any other error is returned, a more generalised error message is presented.

When the sound has finished playing, ASynchSoundLib advises the application by setting the application's "attention" flag to true. Recall that this will cause the AsynchSoundLib function AS_CloseChannel to be called to free up the relevant ASStructure, close the relevant sound channel, clear the "attention" flag, and draw some text in the group box to the right of the image well to indicate to the user that AS_CloseChannel has just been called.

doSpeakStringAsync

doSpeakStringAsync is identical to the function doSpeakStringSync except that, in this function, SpeechBusy is not used to delay the function returning until the speech activity spawned by SpeakString has run its course.

doSetUpDialog

Within doSetUpDialog, the Record Sound Resource bevel button is disabled if the program is running on OS X.