This chapter describes the Sound Input Manager, the part of the Macintosh system software that controls the recording of sound through sound input devices. You can use the Sound Input Manager to display and manage the sound recording dialog box. This ensures that the user is presented with a consistent and standard user interface for sound recording. You can, however, also use Sound Input Manager routines to record sound without the sound recording dialog box or to interact directly with a sound input device driver.

To use this chapter, you should already be familiar with the information in the chapter "Introduction to Sound on the Macintosh" earlier in this book, and in particular with the portions of that chapter that concern sound recording. That chapter explains how your application can record either a sound resource or a sound file using the standard sound recording dialog box. You need to read this chapter only if you need to interact with the Sound Input Manager at a lower level than is allowed by the high-level functions `SndRecord` and `SndRecordToFile`. For example, you need to read this chapter to learn how to

- record sound without using the sound recording dialog box
- interact with a sound input device driver
- write a sound input device driver

To use this chapter, you should also be familiar with the chapter "Sound Manager" in this book, especially the portions of that chapter that describe

- the format of sampled-sound data
- the Macintosh Audio Compression and Expansion (MACE) routines
- the structure of sound resources and sound files
- the use of the `Gestalt` function to determine whether certain sound-related facilities are available.

If you are writing a sound input device driver, you should already be familiar with writing device drivers in general, as described in the book *Inside Macintosh: Devices*.

About the Sound Input Manager

The Sound Input Manager uses sound input device drivers to allow applications to access sound input hardware in a device-independent way. A **sound input device driver** is a standard Macintosh device driver used to interface to an audio digitizer or other recording hardware. If you use the Sound Input Manager's high-level routines, the Sound Input Manager handles all communication with a sound input device driver for you. If, however, you need to use the Sound Input Manager's low-level routines, you must open a sound input device driver yourself. You might also need to get information about certain attributes of a sound input device. Sound input device drivers allow your application to query a device about such attributes.

Sound Recording Without the Standard Interface

The Sound Input Manager provides your application with the ability to record and digitally store sounds in a device-independent manner even if your application does not use the standard sound recording interface. In cases where you need very fine control over the recording process, you can call various low-level sound input routines.

Your application can obtain control over sound recording in two different ways. First, if your application uses the sound recording dialog box, you can modify the dialog box's features by defining a custom filter procedure, as explained in detail in the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*. Second, if your application needs to fine tune the sound recording process itself (or if your application does not use the standard sound recording dialog box), then the application must use the Sound Input Manager's low-level routines.

In instances where you need to gain greater control over the recording process, you can use a set of routines that manipulate the incoming sound data by using sound parameter blocks. The parameter blocks contain information about the current recording device, the length recorded, a routine to call on completion of the recording, and so forth. You can call the `SPBRecord` function (or the `SPBRecordToFile` function) to begin a recording. Then you can use the functions `SPBPauseRecording`, `SPBResumeRecording`, and `SPBStopRecording` to control the recording. Note that you need to open a device (using the `SPBOpenDevice` function) before you can record from it. On completion of the recording, you should close the device (using the `SPBCloseDevice` function).

If you do record sounds using the Sound Input Manager's low-level routines, you also need to set up your own sound resource headers or sound files, because the Sound Input Manager's low-level routines return raw sampled-sound data to your application. The Sound Input Manager provides two functions, `SetupSndHeader` and `SetupAIFFHeader`, that allow you to set up your own sound resource headers or sound files.

Interaction With Sound Input Devices

The Sound Input Manager provides routines that allow your application to request information about a sound input device or to change a sound input device's settings. The types of information you can obtain about a sound input device include

- the name, icon, and icon mask of the device driver
- whether the device driver supports asynchronous recording
- the device's settings, such as the number of channels the device is to record, the compression type, the number of bytes per sample at the current compression setting, and the sample rate to be produced by the device
- the range of compression types, sample rates, and sample sizes that the device supports

You can also use the Sound Input Manager to change some of a sound input device's settings and to turn features on and off. For example, you can turn on and off automatic gain control on some device drivers. **Automatic gain control** moderates sound recording

to give a consistent signal level. Second, you can turn on and off the **playthrough feature,** which allows the user to hear through the Macintosh speaker the sound being recorded. Third, you can turn on and off **VOX recording,** or voice-activated recording, which allows your application to record only when the amplitude of sound input exceeds a certain level. You can use VOX recording either to prevent recording from starting until sound is at least a certain amplitude or to automatically stop recording when sound falls below a certain amplitude. This latter capability is called **VOX stopping**.

An important feature of sound input devices is **continuous recording.** All sound input devices that support asynchronous recording should support continuous recording as well. Continuous recording allows your application to make several consecutive calls to the `SPBRecord` function without losing data between calls. For example, you might need to record a lengthy sound to disk but not be able to fit the entire sound into RAM. Thus, it's important to be able to save a buffer of data to disk while the sound input device driver continues to collect recorded data. The Sound Input Manager's `SndRecordToFile` function relies on continuous recording.

To get information about a device or to turn features on and off, you can use the `SPBGetDeviceInfo` and `SPBSetDeviceInfo` functions. These functions allow you to use sound input device information selectors to specify what type of information you need to know about the device or what settings you wish to change.

Sound Input Device Drivers

The Sound Input Manager also provides several routines intended for use only by sound input device drivers. Sound input device drivers need to register themselves with the Sound Input Manager by calling the `SPBSignInDevice` function. This makes that device visible in the Sound In control panel for possible selection as the current input device. You can remove a device from that panel by calling the `SPBSignOutDevice` function.

For Macintosh computers with built-in sound recording hardware, the system software includes a sound input device driver. This driver automatically calls `SPBSignInDevice` when the computer starts up. If you are creating a sound input device driver for some other sound recording hardware, your device driver must register itself at startup time. Once your driver is registered, it must respond to Status, Control, and Read calls issued by the Sound Input Manager. The Sound Input Manager issues Status calls to get information about a device, Control calls to set device settings, and Read calls to initiate recording.

Using the Sound Input Manager

You can use the Sound Input Manager to record sounds with the sound recording dialog box, to record sounds directly from a device, to get and set information about a sound input device, and to register your sound input device driver so that it can respond to

Read, Status, and Control calls. This section does not explain how to record sounds using the sound recording dialog box; for information on that, see the chapter "Introduction to Sound on the Macintosh" in this book.

Recording Sounds Directly From a Device

The Sound Input Manager provides a number of routines that you can use for low-level control over the recording process (such as the ability to intercept sound input data at interrupt time). You can open a sound input device and read data from it by calling these low-level Sound Input Manager routines. Several of those routines access information through a **sound input parameter block,** which is defined by the `SPB` data type:

```
TYPE SPB =
RECORD
   inRefNum:            LongInt;    {reference number of input device}
   count:               LongInt;    {number of bytes to record}
   milliseconds:        LongInt;    {number of milliseconds to record}
   bufferLength:        LongInt;    {length of buffer to record into}
   bufferPtr:           Ptr;        {pointer to buffer to record into}
   completionRoutine:   ProcPtr;    {pointer to a completion routine}
   interruptRoutine:    ProcPtr;    {pointer to an interrupt routine}
   userLong:            LongInt;    {for application's use}
   error:               OSErr;      {error returned after recording}
   unused1:             LongInt;    {reserved}
END;
```

The `inRefNum` field indicates the reference number of the sound input device from which the recording is to occur. You can obtain the reference number of the default sound input device by using the `SPBOpenDevice` function.

The `count`, `milliseconds`, and `bufferLength` fields jointly determine the length of recording. The `count` field indicates the number of bytes to record; the `milliseconds` field indicates the number of milliseconds to record; and the `bufferLength` field indicates the length in bytes of the buffer into which the recorded sound data is to be placed. If the `count` and `milliseconds` fields are not equivalent, then the field which specifies the longer recording time is used. If the buffer specified by the `bufferLength` field is shorter than this recording time, then the recording time is truncated so that the recorded data can fit into the buffer specified by the `bufferPtr` field. The Sound Input Manager provides two functions, `SPBMilliSecondsToBytes` and `SPBBytesToMilliSeconds`, that allow you to convert between byte and millisecond values.

After recording finishes, the `count` and `milliseconds` fields indicate the number of bytes and milliseconds actually recorded.

The `completionRoutine` and `interruptRoutine` fields allow your application to define a sound input completion routine and a sound input interrupt routine, respectively. More information on these routines is provided later in this section.

The `userLong` field contains a long integer that is provided for your application's own use. You can use this field, for instance, to pass a handle to an application-defined structure to the sound input completion or interrupt routine. Or, you can use this field to store the value of your application's A5 register, so that your sound input completion or interrupt routine can access your application's global variables. For more information on preserving the value of the A5 register, see the discussion of the `SetA5` and `SetCurrentA5` functions in the chapter "Memory Management Utilities" in *Inside Macintosh: Memory*.

The `error` field describes any errors that occur during the recording. This field contains a value greater than 0 while recording unless an error occurs, in which case it contains a value less than 0 that indicates an operating system error. Your application can poll this field to check on the status of an asynchronous recording. If recording terminates without an error, this field contains 0.

Listing 3-1 shows how to set up a sound parameter block and record synchronously using the `SPBRecord` function. This procedure takes one parameter, a handle to a block of memory in which the recorded sound data is to be stored. It is assumed that the block of memory is large enough to hold the sound to be recorded.

Listing 3-1 Recording directly from a sound input device

```
PROCEDURE MyRecordSnd (mySndH: Handle);
CONST
   kAsync = TRUE;
   kMiddleC = 60;
VAR
   mySPB:         SPB;        {a sound input parameter block}
   myInRefNum:    LongInt;    {device reference number}
   myBuffSize:    LongInt;    {size of buffer to record into}
   myHeadrLen:    Integer;    {length of sound header}
   myNumChans:    Integer;    {number of channels}
   mySampSize:    Integer;    {size of a sample}
   mySampRate:    Fixed;      {sample rate}
   myCompType:    OSType;     {compression type}
   myErr:         OSErr;
BEGIN
   {Open the default input device for reading and writing.}
   myErr := SPBOpenDevice('', siWritePermission, myInRefNum);

   IF myErr = noErr THEN
   BEGIN
      {Get current settings of sound input device.}
      MyGetDeviceSettings(myInRefNum, myNumChans, mySampRate,
                          mySampSize, myCompType);
```

```
      {Set up handle to contain the 'snd ' resource header.}
      myErr := SetupSndHeader(mySndH, myNumChans, mySampRate,mySampSize,
                              myCompType, kMiddleC, 0, myHeadrLen);

      {Leave room in buffer for the sound resource header.}
      myBuffSize := GetHandleSize(mySndH) - myHeadrLen;

      {Lock down the sound handle until the recording is over.}
      HLockHi(mySndH);

      {Set up the sound input parameter block.}
      WITH mySPB do
      BEGIN
         inRefNum := myInRefNum;          {input device reference number}
         count := myBuffSize;             {number of bytes to record}
         milliseconds := 0;               {no milliseconds}
         bufferLength := myBuffSize;      {length of buffer}
         bufferPtr := Ptr(ORD4(mySndH^) + myHeadrLen);
                                          {put data after 'snd ' header}
         completionRoutine := NIL;        {no completion routine}
         interruptRoutine := NIL;         {no interrupt routine}
         userLong := 0;                   {no user data}
         error := noErr;                  {clear error field}
         unused1 := 0;                    {clear reserved field}
      END;

      {Record synchronously through the open sound input device.}
      myErr := SPBRecord(@mySPB, NOT kAsync);

      HUnlock(mySndH);                    {unlock the handle}

      {Indicate the number of bytes actually recorded.}
      myErr := SetupSndHeader(mySndH, myNumChans, mySampRate, mySampSize,
                              myCompType, kMiddleC, mySPB.count,
                              myHeadrLen);

      {Close the input device.}
      myErr := SPBCloseDevice(myInRefNum);
   END;
END;
```

The MyRecordSnd procedure defined in Listing 3-1 opens the default sound input device by using the SPBOpenDevice function. You can specify one of two values for the permission parameter of SPBOpenDevice:

```
CONST
   siReadPermission  = 0;  {open device for reading}
   siWritePermission = 1;  {open device for reading/writing}
```

You must open a device for both reading and writing if you intend to use the SPBSetDeviceInfo function or the SPBRecord function. If SPBOpenDevice successfully opens the specified device for reading and writing, MyRecordSnd calls the MyGetDeviceSettings procedure (defined in Listing 3-3 on page 3-12). That procedure calls the Sound Input Manager function SPBGetDeviceInfo (explained in "Getting and Setting Sound Input Device Information" on page 3-10) to determine the current number of channels, sample rate, sample size, and compression type in use by the device.

This information is then passed to the SetupSndHeader function, which sets up the handle mySndH with a sound header describing the current device settings. After doing this, MyRecordSnd sets up a sound input parameter block and calls the SPBRecord function to record a sound. Note that the handle must be locked during the recording because the parameter block contains a pointer to the input buffer. After the recording is done, MyRecordSnd once again calls the SetupSndHeader function to fill in the actual number of bytes recorded.

If the MyRecordSnd procedure defined in Listing 3-1 executes successfully, the handle mySndH points to a resource of type 'snd '. Your application can then synchronously play the recorded sound, for example, by executing the following line of code:

```
myErr := SndPlay(NIL, mySndH, FALSE);
```

For more information on playing sounds your application has recorded, see the chapter "Sound Manager" in this book.

Defining a Sound Input Completion Routine

The completionRoutine field of the sound parameter block record contains the address of a completion routine that is executed when the recording terminates normally, either by reaching its prescribed time or size limits or by the application calling the SPBStopRecording function. A completion routine should have the following format:

```
PROCEDURE MySICompletionRoutine (inParamPtr: SPBPtr);
```

The completion routine is passed the address of the sound input parameter block that was passed to the SPBRecord function. You can gain access to other data structures in your application by passing an address in the userLong field of the parameter block. After the completion routine executes, your application should check the error field of the sound input parameter block to see if an error code was returned.

Your sound input interrupt routine is always called at interrupt time, so it should not call routines that might allocate or move memory or assume that A5 is set up. For more information on sound input interrupt routines, see "Sound Input Interrupt Routines" beginning on page 3-55.

Defining a Sound Input Interrupt Routine

The `interruptRoutine` field of the sound input parameter block contains the address of a routine that executes when the internal buffers of an asynchronous recording device are filled. The internal buffers contain raw sound samples taken directly from the input device. The interrupt routine can modify the samples in the buffer in any way it requires. The processed samples are then written to the application buffer. If compression is enabled, the modified data is compressed after your interrupt routine operates on the samples and before the samples are written to the application buffer.

Your sound input interrupt routine is always called at interrupt time, so it should not call routines that might allocate or move memory or assume that A5 is set up. For more information on sound input interrupt routines, see "Sound Input Interrupt Routines" beginning on page 3-55.

Getting and Setting Sound Input Device Information

You can get information about a specific sound input device and alter a sound input device's settings by calling the functions `SPBGetDeviceInfo` and `SPBSetDeviceInfo`. These functions accept **sound input device information selectors** that determine which information you need or want to change. The selectors currently available are defined by constants of type `OSType`.

Here is a list of the selectors that all sound input device drivers must support. For complete details on all the selectors described in this section, see "Sound Input Device Information Selectors" beginning on page 3-18.

```
CONST
   siAsync                 = 'asyn';   {asynchronous capability}
   siChannelAvailable      = 'chav';   {number of channels available}
   siCompressionAvailable  = 'cmav';   {compression types available}
   siCompressionFactor     = 'cmfa';   {current compression factor}
   siCompressionType       = 'comp';   {compression type}
   siContinuous            = 'cont';   {continuous recording}
   siDeviceBufferInfo      = 'dbin';   {size of interrupt buffer}
   siDeviceConnected       = 'dcon';   {input device connection status}
   siDeviceIcon            = 'icon';   {input device icon}
   siDeviceName            = 'name';   {input device name}
   siLevelMeterOnOff       = 'lmet';   {level meter state}
   siNumberChannels        = 'chan';   {current number of channels}
   siRecordingQuality      = 'qual';   {recording quality}
   siSampleRate            = 'srat';   {current sample rate}
```

```
   siSampleRateAvailable   = 'srav';   {sample rates available}
   siSampleSizeAvailable   = 'ssav';   {sample sizes available}
   siSampleSize            = 'ssiz';   {current sample size}
   siTwosComplementOnOff   = 'twos';   {two's complement state}
```

The Sound Input Manager defines several selectors that specifically help it interact with sound input device drivers. Your application should not use any of these selectors, but if you are implementing a sound input device driver, you need to support these selectors. They are:

```
CONST
   siCloseDriver           = 'clos';   {release driver}
   siInitializeDriver      = 'init';   {initialize driver}
   siPauseRecording        = 'paus';   {pause recording}
   siUserInterruptProc     = 'user';   {set sound input interrupt routine}
```

Finally, there are a number of sound input device information selectors that sound input device drivers can optionally support. If you are writing an application, you can use these selectors to interact with a sound input device driver, but you should be aware that some drivers might not support all of them. To determine if a driver supports one of these selectors, you can use the `SPBGetDeviceInfo` function. If no errors are returned, then the selector is supported when using the `SPBGetDeviceInfo` and the `SPBSetDeviceInfo` functions.

```
CONST
   siActiveChannels        = 'chac';   {channels active}
   siActiveLevels          = 'lmac';   {levels active}
   siAGCOnOff              = 'agc ';   {automatic gain control state}
   siCompressionHeader     = 'cmhd';   {get compression header}
   siCompressionNames      = 'cnam';   {return compression type names}
   siInputGain             = 'gain';   {input gain level}
   siInputSource           = 'sour';   {input source selector}
   siInputSourceNames      = 'snam';   {input source names}
   siOptionsDialog         = 'optd';   {display options dialog box}
   siPlayThruOnOff         = 'plth';   {play-through state}
   siStereoInputGain       = 'sgai';   {stereo input gain level}
   siVoxRecordInfo         = 'voxr';   {VOX record parameters}
   siVoxStopInfo           = 'voxs';   {VOX stop parameters}
```

The format of the relevant data (either returned by the Sound Input Manager or provided by you) depends on the selector you provide. For example, if you want to determine the name of some sound input device, you can pass to the `SPBGetDeviceInfo` function the `siDeviceName` selector and a pointer to a 256-byte buffer. If the `SPBGetDeviceInfo` function can get the information, it fills that buffer with the name of the specified sound input device. Listing 3-2 illustrates one way you can determine the name of a particular sound input device.

Listing 3-2 Determining the name of a sound input device

```
FUNCTION MyGetDeviceName (myRefNum: LongInt; VAR dName: Str255): OSErr;
BEGIN
   MyGetDeviceName := SPBGetDeviceInfo(myRefNum, siDeviceName, Ptr(@dName));
END;
```

Note
You can get the name and icon of all connected sound input devices without using sound input information selectors by using the `SPBGetIndexedDevice` function, which is described on page 3-49. ◆

Some selectors cause the `SPBGetDeviceInfo` function to return data of other types. Listing 3-3 illustrates how to determine the number of channels, the sample rate, the sample size, and the compression type currently in use by a given sound input device. (The procedure defined in Listing 3-3 is called in the procedure defined in Listing 3-1.)

Listing 3-3 Determining some sound input device settings

```
PROCEDURE MyGetDeviceSettings (myRefNum: LongInt;
                               VAR numChannels: Integer;
                               VAR sampleRate: Fixed;
                               VAR sampleSize: Integer;
                               VAR compressionType: OSType);
VAR
   myErr:      OSErr;
BEGIN
   {Get number of active channels.}
   myErr := SPBGetDeviceInfo (myRefNum, siNumberChannels, Ptr(@numChannels));
   {Get sample rate.}
   myErr := SPBGetDeviceInfo(myRefNum, siSampleRate, Ptr(@sampleRate));
   {Get sample size.}
   myErr := SPBGetDeviceInfo(myRefNum, siSampleSize, Ptr(@sampleSize));
   {Get compression type.}
   myErr := SPBGetDeviceInfo(myRefNum, siCompressionType,
                                                   Ptr(@compressionType));
END;
```

All of the selectors that return a handle allocate the memory for that handle in the current heap zone; you are responsible for disposing of that handle when you are done with it, and you should verify that there is enough memory for such a handle before calling the selector.

Writing a Sound Input Device Driver

This section describes what you need to do when you do write a sound input device driver. If you write a sound input device driver, you should set the `drvrFlags` field of the sound input device driver's header to indicate that the driver can handle Status, Control, and Read requests. The driver header should also indicate that the driver needs to be locked.

IMPORTANT
You don't need to write a device driver to use sound input capabilities. ▲

After you create a device driver, you must write an extension that installs it. Before your extension installs the driver, it should pass the `Gestalt` function the `gestaltSoundAttr` attribute selector and inspect the `gestaltSoundIOMgrPresent` bit to determine if the sound input routines are available. If so, the extension should install the sound input device driver into the unit table just as any other driver must be installed.

After installing the driver, the extension must then make an Open request to the driver, so that the driver can perform any necessary initialization. In particular, the driver might set the `dCtlStorage` field of the device control entry to a pointer or a handle to a block in the system heap containing all of the variables that it might need. Finally, the device driver signs into the Sound Input Manager by calling the `SPBSignInDevice` function.

Once signed in, a driver can receive Status, Control, and Read requests from the Sound Input Manager. On entry, the A0 register contains a pointer to a standard Device Manager parameter block, and the A1 register contains a pointer to the device control entry. For more information on using registers in a device driver, see *Inside Macintosh: Devices*.

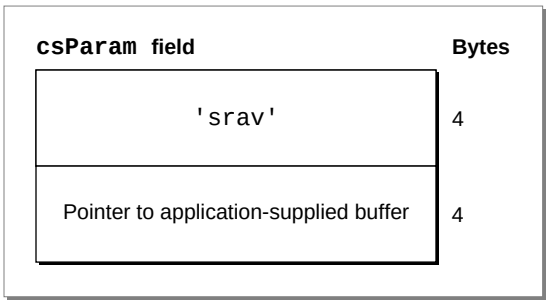
Responding to Status and Control Requests

The Sound Input Manager supports sound input device information selectors by sending your device driver Status and Control requests. It uses Status requests to get information about your device; it uses Control requests to change settings of your sound input device.

The behavior of your sound input device driver in response to Status and Control requests depends on the value of the `csCode` field of the Device Manager control parameter block. If the `csCode` field contains 2, then the sound input information selector is passed in the first 4 bytes of the `csParam` field of the Device Manager control parameter block. For Status requests, the next 18 bytes can be used for your device driver to pass information back to an application. For Control requests, these 18 bytes are used by an application to pass data to your sound input device driver.

Figure 3-1 shows the contents of the `csParam` field of the Device Manager control parameter block for a sample Status request. The first four bytes of the `csParam` field contain the input selector `'srav'`, which is a request for the available sample rates. The next four bytes of the field contain a pointer to an application-supplied buffer in which to return the data (the number of rates available) from the Status request.

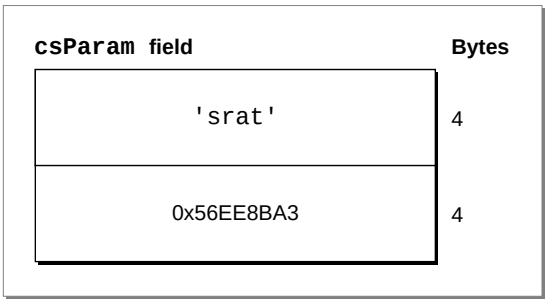
Figure 3-1 An example of the `csParam` field for a Status request



On exit from the Status request, your sound input device driver can respond in one of two ways. If you are returning fewer than 18 bytes of data, your device driver should specify in the first 4 bytes of the `csParam` field of the Device Manager control parameter block the number of bytes of data being returned and place the data in the following 18 bytes. In this case, the Sound Input Manager copies the data to the application-supplied buffer identified in Figure 3-1. If you are returning more than 18 bytes of data, your device driver should copy the data to the application-supplied buffer. In this case, your device driver needs to place a zero in the first 4 bytes of the `csParam` field to indicate to the Sound Input Manager that the data has already been copied to the application-supplied buffer.

Figure 3-2 shows the contents of the `csParam` field of the Device Manager control parameter block for a sample Control request. The first four bytes of the `csParam` field contain the input selector `'srat'` which determines the sample rate for the sound input device. The next eighteen bytes contain the data, which in this example is the sample rate to set for your sound input device. This is a `Fixed` value of four bytes in length.

Figure 3-2 An example of the `csParam` field for a Control request



Note
Some sound input information selectors require your sound input device driver to allocate a handle in which to store information. In this case, your driver should attempt to allocate an appropriately sized handle in the current heap zone. If allocation fails, your driver should return the appropriate Memory Manager result code. ◆

Your sound input device driver must respond to a core set of selectors, but the remaining selectors defined by Apple are optional. Your device driver might also define private selectors to support proprietary features. (Selectors containing all lowercase letters, however, are reserved by Apple.) The section "Getting and Setting Sound Input Device Information" beginning on page 3-10 lists the core selectors and other selectors that have been defined.

If the `csCode` field contains 1 (which can occur only for Control requests), the Sound Input Manager is attempting to stop asynchronous recording; that is, it is issuing a `KillIO` request. In response to this, the driver should stop copying data to the application buffer, update the `ioActCount` field of the request parameter block, and return via an `RTS` instruction.

Before exiting after a Status and Control request, your sound input device driver should fill the D0 register with the appropriate result code or `noErr`. To exit, your sound input device driver should check whether the Status and Control request was executed immediately or was queued.

Note
In current versions of system software, the Sound Input Manager always issues Status and Control requests immediately. This might change in future versions of system software. ◆

Your sound input device driver can determine whether a request is issued immediately by checking the `noQueueBit` in the `ioTrap` field of the Device Manager control parameter block. If the request was made immediately, the Control routine should return via an `RTS` instruction; if the request was queued, the Control routine should jump to the Device Manager's `IODone` function via the global jump vector `JIODone`. You need to make sure that the A0 and A1 registers are set the same as they are on entry to the device driver or `JIODone` will fail.

Responding to Read Requests

When a sound input device receives a Read request, it must start recording and saving recorded data into the buffer specified by the `ioBuffer` field of the request parameter block. If that field is `NIL`, the driver should record but not save the data. During a Read request, your sound input device driver can access the sound parameter block that initiated recording through the `ioMisc` field of the request parameter block.

If a previous Control request has assigned a sound input interrupt routine to the device driver and your driver records asynchronously, then the driver must call the routine each time its internal buffer becomes filled, setting up registers as described in "Defining a Sound Input Interrupt Routine" on page 3-10. The buffer size that your device driver specifies in the D1 register should indicate how much your device records during every

interrupt. For example, a sound input device driver that uses the serial port might use a buffer as small as 3 bytes. For the built-in sound input port on the Macintosh LC and other Macintosh models, the buffer is 512 bytes long.

Your device driver should update the `ioActCount` field of the request parameter block with the actual number of bytes of sampled-sound data recorded. This allows the Sound Input Manager to monitor the activity of your device driver. Whether your device driver operates synchronously or asynchronously, it should complete recording by jumping to the Device Manager's `IODone` function via the global jump vector `JIODone`. You need to set the D0 register to the appropriate result code before jumping to the Device Manager's `IODone` function.

Supporting Stereo Recording

Many sound input devices support recording stereo sounds (that is, sounds from two or more channels). If you are writing a device driver for a stereo device, you need to make sure that you support the `siNumberChannels`, `siActiveChannels`, and `siActiveLevels` selectors.

The `siNumberChannels` selector controls the number of sound input channels and thereby determines the format of the data stream your device driver produces. If the number of channels is 1, the driver should produce monophonic data in response to a Read request. If the number of channels is 2, the driver should produce interleaved stereo data in response to a Read request.

The `siActiveChannels` selector controls which of the available input channels are used for recording. The active channels are specified using a bitmap value. For example, the value $01 indicates that the first channel (the left channel) is to be used. The value $02 indicates that the second channel (the right channel) is to be used.

The `siNumberChannels` and `siActiveChannels` selectors together determine the exact format of the output data stream. If the current number of channels is 1 and the current active channel bitmap is $01, the driver should produce a stream of monophonic data containing samples only from the left input channel. If the current number of channels is 1 and the current active channel bitmap is $02, the driver should produce a stream of monophonic data containing samples only from the right input channel. If the current number of channels is 1 and the current active channel bitmap is $03, the driver should mix the right and left channels to produce a stream of monophonic data. If the current number of channels is 2 and the current active channel bitmap is $03, the driver should produce a stream of interleaved samples from the left and right input channels.

Note
If the `siActiveChannels` selector is never passed to a sound input device driver, it's recommended that the active channel default bitmap for both monophonic and stereo recording should be $03. When the active channel bitmap conflicts with the number of channels (for example, there are two channels but the active channel bitmap is $01), you should use the default value of $03. ◆

Supporting Continuous Recording

If your sound input device driver supports continuous recording, it must do more than respond to Status, Control, and Read requests. It must also, if continuous recording is on, begin recording into an internal ring buffer as soon as a Read request completes. The buffer should be made large enough so that the sound input device driver can support successive requests to the `SPBRecord` function in most circumstances; however, if your driver exhausts the internal buffer, your driver should begin recording again at the start of the buffer.

When the sound input device driver receives a subsequent Read request, it should record to the application's buffer first all of the data in the internal ring buffer and then as much fresh data as it can record during one interrupt.

If a Read terminates due to a `KillIO` request, your sound input device driver does not need to continue recording samples to the internal ring buffer until after the next uninterrupted Read request.

Sound Input Manager Reference

This section describes the constants, data structure, and the routines provided by the Sound Input Manager.

Constants

This section describes the constants you can use with the `SPBSetDeviceInfo` and `SPBGetDeviceInfo` functions to set or get device information. It also lists the `Gestalt` function sound attributes selector and the returned bit numbers that are relevant to the Sound Input Manager. All other constants defined by the Sound Input Manager are described at the appropriate location in this chapter. (For example, the constants that you can use to specify sound recording qualities are described in connection with the `SndRecord` function beginning on page 3-28.)

Gestalt Selector and Response Bits

You can pass the `gestaltSoundAttr` selector to the `Gestalt` function to determine information about the sound input capabilities of a Macintosh computer.

```
CONST
   gestaltSoundAttr           = 'snd ';   {sound attributes selector}
```

The `Gestalt` function returns information by setting or clearing bits in the `response` parameter. The bits relevant to the Sound Input Manager are defined by constants:

```
CONST
   gestaltSoundIOMgrPresent   = 3;     {sound input routines available}
   gestaltBuiltInSoundInput   = 4;     {built-in input hw available}
   gestaltHasSoundInputDevice = 5;     {sound input device available}
   gestaltPlayAndRecord       = 6;     {built-in hw can play while recording}
   gestalt16BitSoundIO        = 7;     {built-in hw can handle 16-bit data}
   gestaltStereoInput         = 8;     {built-in hw can record stereo sounds}
   gestaltLineLevelInput      = 9;     {built-in input hw needs line level}
```

Constant descriptions

`gestaltSoundIOMgrPresent`
Set if the Sound Input Manager is available.

`gestaltBuiltInSoundInput`
Set if a built-in sound input device is available.

`gestaltHasSoundInputDevice`
Set if a sound input device is available. This device can be either built-in or external.

`gestaltPlayAndRecord`
Set if the built-in sound hardware is able to play and record sounds simultaneously. If this bit is clear, the built-in sound hardware can either play or record, but not do both at once. This bit is valid only if the `gestaltBuiltInSoundInput` bit is set, and it applies only to any built-in sound input and output hardware.

`gestalt16BitSoundIO`
Set if the built-in sound hardware is able to play and record 16-bit samples. This indicates that built-in hardware necessary to handle 16-bit data is available.

`gestaltStereoInput`
Set if the built-in sound hardware can record stereo sounds.

`gestaltLineLevelInput`
Set if the built-in sound input port requires line level input.

Note

For complete information about the `Gestalt` function, see the chapter "Gestalt Manager" in *Inside Macintosh: Operating System Utilities*. ◆

Sound Input Device Information Selectors

You can call the `SPBSetDeviceInfo` and `SPBGetDeviceInfo` functions to set or get information about a sound input device. You pass each of those functions a sound input device information selector in the `infoType` parameter to specify the type of information you need. The available device information selectors are defined by constants.

IMPORTANT

Some of these selectors are intended for use only by the Sound Input Manager and other parts of the system software that need to interact directly with sound input device drivers. (For example, the Sound Input Manager sends the `siCloseDriver` selector to a sound input device driver when it is closing the device.) In general, applications should not use these reserved selectors. ▲

```
CONST
   siActiveChannels        = 'chac';   {channels active}
   siActiveLevels          = 'lmac';   {levels active}
   siAGCOnOff              = 'agc ';   {automatic gain control state}
   siAsync                 = 'asyn';   {asynchronous capability}
   siChannelAvailable      = 'chav';   {number of channels available}
   siCloseDriver           = 'clos';   {reserved for internal use only}
   siCompressionAvailable  = 'cmav';   {compression types available}
   siCompressionFactor     = 'cmfa';   {current compression factor}
   siCompressionHeader     = 'cmhd';   {return compression header}
   siCompressionNames      = 'cnam';   {return compression type names}
   siCompressionType       = 'comp';   {current compression type}
   siContinuous            = 'cont';   {continuous recording}
   siDeviceBufferInfo      = 'dbin';   {size of interrupt buffer}
   siDeviceConnected       = 'dcon';   {input device connection status}
   siDeviceIcon            = 'icon';   {input device icon}
   siDeviceName            = 'name';   {input device name}
   siInitializeDriver      = 'init';   {reserved for internal use only}
   siInputGain             = 'gain';   {input gain level}
   siInputSource           = 'sour';   {input source selector}
   siInputSourceNames      = 'snam';   {input source names}
   siLevelMeterOnOff       = 'lmet';   {level meter state}
   siNumberChannels        = 'chan';   {current number of channels}
   siOptionsDialog         = 'optd';   {display options dialog box}
   siPauseRecording        = 'paus';   {reserved for internal use only}
   siPlayThruOnOff         = 'plth';   {play-through state}
   siRecordingQuality      = 'qual';   {recording quality}
   siSampleRate            = 'srat';   {current sample rate}
   siSampleRateAvailable   = 'srav';   {sample rates available}
   siSampleSize            = 'ssiz';   {current sample size}
   siSampleSizeAvailable   = 'ssav';   {sample sizes available}
   siStereoInputGain       = 'sgai';   {stereo input gain level}
   siTwosComplementOnOff   = 'twos';   {two's complement state}
   siUserInterruptProc     = 'user';   {reserved for internal use only}
   siVoxRecordInfo         = 'voxr';   {VOX record parameters}
   siVoxStopInfo           = 'voxs';   {VOX stop parameters}
```

Constant descriptions

`siActiveChannels`
: Get or set the channels to record from. When setting the active channels, the data passed in is a long integer that is interpreted as a bitmap describing the channels to record from. For example, if bit 0 is set, then the first channel is made active. The samples for each active channel are interleaved in the application's buffer. When reading the active channels, the data returned is a bitmap of the active channels.

`siActiveLevels`
: Get the current signal level for each active channel. The `infoData` parameter points to an array of integers, the size of which depends on the number of active channels. You can determine how many channels are active by calling `SPBGetDeviceInfo` with the `siNumberChannels` selector.

`siAGCOnOff`
: Get or set the current state of the automatic gain control feature. The `infoData` parameter points to an integer, which is 0 if gain control is off and 1 if it is on.

`siAsync`
: Determine whether the driver supports asynchronous recording functions. The `infoData` parameter points to an integer, which is 0 if the driver supports synchronous calls only and 1 otherwise. Some sound input drivers do not support asynchronous recording at all, and some might support asynchronous recording only on certain hardware configurations.

`siChannelAvailable`
: Get the maximum number of channels this device can record. The `infoData` parameter points to an integer, which is the number of available channels.

`siCloseDriver`
: The Sound Input Manager sends this selector when it closes a device previously opened with write permission. The sound input device driver should stop any recording in progress, deallocate the input hardware, and initialize local variables to default settings. Your application should never issue this selector directly. The `infoData` parameter is unused with this selector.

`siCompressionAvailable`
: Get the number and list of compression types this device can produce. The `infoData` parameter points to an integer, which is the number of compression types, followed by a handle. The handle references a list of compression types, each of type `OSType`.

`siCompressionFactor`
: Get the compression factor of the current compression type. For example, the compression factor for MACE 3:1 compression is 3. If a sound input device driver supports only compression type `'NONE'`, the returned compression type is 1. The `infoData` parameter points to an integer, which is the compression factor.

`siCompressionHeader`
: Get a compressed sound header for the current recording settings. Your application passes in a pointer to a compressed sound header

and the driver fills it in. Before calling `SPBGetDeviceInfo` with this selector, you should set the `numFrames` field of the compressed sound header to the number of bytes in the sound. When `SPBGetDeviceInfo` returns successfully, that field contains the number of sample frames in the sound. This selector is needed only by drivers that use compression types that are not directly supported by Apple. If you call this selector after recording a sound, your application can get enough information about the sound to play it or save it in a file. The `infoData` parameter points to a compressed sound header.

`siCompressionNames`
Get a list of names of the compression types supported by the sound input device. In response to a Status call, a sound input device driver returns, in the location specified by the `infoData` parameter, a handle to a block of memory that contains the names of all compression types supported by the driver. It is the driver's responsibility to allocate that block of memory, but it should not release it. The software issuing this selector is responsible for disposing of the handle. As a result, a device driver must detach any resource handles (by calling `DetachResource`) before returning them to the caller. The data in the handle has the same format as an `'STR#'` resource: a two-byte count of the strings in the resource, followed by the strings themselves. The strings should occur in the same order as the compression types returned by the `siCompressionAvailable` selector. If the driver does not support compression, it returns `siUnknownInfoType`. If the driver supports compression but for some reason not all compression types are currently selectable, it returns a list of all available compression types.

`siCompressionType`
Get or set the compression type. Some devices allow the incoming samples to be compressed before being placed in your application's input buffer. The `infoData` parameter points to a buffer of type `OSType`, which is the compression type.

`siContinuous` Get or set the state of continuous recording from this device. If recording is being turned off, the driver stops recording samples to its internal buffer. Only sound input device drivers that support asynchronous recording support continuous recording. The `infoData` parameter points to an integer, which is the state of continuous recording (0 is off, 1 is on).

`siDeviceBufferInfo`
Get the size of the device's internal buffer. This information can be useful when you want to modify sound input data at interrupt time. Note, however, that if a driver is recording continuously, then the size of the buffer passed to your sound input interrupt routine might be greater than the size this selector returns because data recorded between calls to `SPBRecord` as well as recorded during calls to `SPBRecord` will be sent to your interrupt routine. The `infoData` parameter points to a long integer, which is the size of the device's internal buffer.

`siDeviceConnected`

Get the state of the device connection. The `infoData` parameter points to an integer, which is one of the following constants:

```
CONST
   siDeviceIsConnected     = 1;
   siDeviceNotConnected    = 0;
   siDontKnowIfConnected   = -1;
```

The `siDeviceIsConnected` constant indicates that the device is connected and ready. The `siDeviceNotConnected` constant indicates that the device is not connected. The `siDontKnowIfConnected` constant indicates that the Sound Input Manager cannot determine whether the device is connected.

`siDeviceIcon` Get the device's icon and icon mask. In response to a Status call, a sound input device driver should return, in the location specified by the `infoData` parameter, a handle to a block of memory that contains the icon and its mask in the format of an `'ICN#'` resource. It is the driver's responsibility to allocate that block of memory, but it should not releasee it. The software issuing this selector is responsible for disposing of the handle. As a result, a device driver should detach any resource handles (by calling `DetachResource`) before returning them to the caller.

`siDeviceName` Get the name of the sound input device. Your application must pass a pointer to a buffer that will be filled in with the device's name. The buffer needs to be large enough to hold a `Str255` data type.

`siInitializeDriver`

The Sound Input Manager sends this selector when it opens a sound input device with write permission. The sound input device driver initializes local variables and prepares to start recording. If possible, the driver initializes the device to a sampling rate of 22 kHz, a sample size of 8 bits, mono recording, no compression, automatic gain control on, and all other features off. Your application should never issue this selector directly. The `infoData` parameter is unused with this selector.

`siInputGain` Get and set the current sound input gain. If the available hardware allows adjustment of the recording gain, this selector lets you get and set the gain. In response to a Status call, a sound input driver returns the current gain setting. In response to a Control call, a sound input driver sets the gain level used for all subsequent recording to the specified value. The `infoData` parameter points to a 4-byte value of type `Fixed` ranging from 0.5 to 1.5, where 1.5 specifies maximum gain.

`siInputSource` Get and set the current sound input source. If the available hardware allows recording from more than one source, this selector lets you get and set the source. In response to a Status call, a sound input driver returns the current source value; if the driver supports only one source, it returns `siUnknownInfoType`. In response to a Control call, a sound input driver sets the source of all subsequent

recording to the value passed in. If the value is less than 1 or greater than the number of input sources, the driver returns `paramErr`; if the driver supports only one source, it returns `siUnknownInfoType`. The `infoData` parameter points to an integer, which is the index of the current sound input source.

`siInputSourceNames`

Get a list of the names of all the sound input sources supported by the sound input device. In response to a Status call, a sound input device driver returns, in the location specified by the `infoData` parameter, a handle to a block of memory that contains the names of all sound sources supported by the driver. It is the driver's responsibility to allocate that block of memory, but it should not release it. The software issuing this selector is responsible for disposing of the handle. As a result, a device driver must detach any resource handles (by calling `DetachResource`) before returning them to the caller. The data in the handle has the same format as an `'STR#'` resource: a two-byte count of the strings in the resource, followed by the strings themselves. The strings should occur in the same order as the input sources returned by the `siInputSource` selector. If the driver supports only one source, it returns `siUnknownInfoType`. If the driver supports more than one source but for some reason not all of them are currently selectable, it returns a list of all available input sources.

`siLevelMeterOnOff`

Get or set the current state of the level meter. For calls to set the level meter, the `infoData` parameter points to an integer that indicates whether the level meter is off (0) or on (1). To get the level meter setting, the `infoData` parameter points to two integers; the first integer indicates the state of the level meter, and the second integer contains the level value of the meter. The level meter setting is an integer that ranges from 0 (no volume) to 255 (full volume).

`siNumberChannels`

Get or set the number of channels this device is to record. The `infoData` parameter points to an integer, which indicates the number of channels. Note that this selector determines the format of the data stream output by the driver. If the number of channels is 1, the driver should output monophonic data in response to a Read call. If the number of channels is 2, the driver should output interleaved stereo data.

`siOptionsDialog`

Determine whether the driver supports an Options dialog box (`SPBGetDeviceInfo`) or cause the driver to display the Options dialog box (`SPBSetDeviceInfo`). This dialog box is designed to allow the user to configure device-specific features of the sound input hardware. With `SPBGetDeviceInfo`, the `infoData` parameter points to an integer, which indicates whether the driver supports an Options dialog box (1 if it supports it, 0 otherwise). With `SPBSetDeviceInfo`, the `infoData` parameter is unused.

`siPauseRecording`

The Sound Input Manager uses this selector to get or set the current pause state. The sound input device driver continues recording but does not store the sampled data in a buffer. Your application should never issue this selector directly. The `infoData` parameter points to an integer, which indicates the state of pausing (0 is off, 1 is on).

`siPlayThruOnOff`

Get or set the current play-through state and volume. The `infoData` parameter points to an integer, which indicates the current play-through volume (1 to 7). If that integer is 0, then play-through is off.

`siRecordingQuality`

Get or set the current quality of recorded sound. The `infoData` parameter points to a buffer of type `OSType`, which is the recording quality. Currently three qualities are supported, defined by these constants:

```
CONST
   siBestQuality           = 'best';
   siBetterQuality         = 'betr';
   siGoodQuality           = 'good';
```

These qualities are defined by the sound input device driver. Usually *best* means monaural, 8-bit, 22 kHz, sound with no compression.

`siSampleRate` Get or set the sample rate to be produced by this device. The sample rate must be in the range 0 to 65535.65535 Hz. The sample rate is declared as a `Fixed` data type. In order to accommodate sample rates greater than 32 kHz, the most significant bit is not treated as a sign bit; instead, that bit is interpreted as having the value 32,768. The `infoData` parameter points to a buffer of type `Fixed`, which is the sample rate.

`siSampleRateAvailable`

Get the range of sample rates this device can produce. The `infoData` parameter points to an integer, which is the number of sample rates the device supports, followed by a handle. The handle references a list of sample rates, each of type `Fixed`. If the device can record a range of sample rates, the number of sample rates is set to 0 and the handle contains two rates, the minimum and the maximum of the range of sample rates. Otherwise, a list is returned that contains the sample rates supported. In order to accommodate sample rates greater than 32 kHz, the most significant bit is not treated as a sign bit; instead, that bit is interpreted as having the value 32,768.

`siSampleSize` Get or set the sample size to be produced by this device. Because some compression formats require specific sample sizes, this selector might return an error when compression is used. The `infoData` parameter points to an integer, which is the sample size.

`siSampleSizeAvailable`
: Get the range of sample sizes this device can produce. The `infoData` parameter points to an integer, which is the number of sample sizes the device supports, followed by a handle. The handle references a list of sample sizes, each of type `Integer`.

`siStereoInputGain`
: Get and set the current stereo sound input gain. If the available hardware allows adjustment of the recording gain, this selector lets you get and set the gain for each of two channels (left or right). In response to a Status call, a sound input driver should return the current gain setting for the specified channel. In response to a Control call, a sound input driver should set the gain level used for all subsequent recording to the specified value. The `infoData` parameter points to two 4-byte values of type `Fixed` ranging from 0.5 to 1.5, where 1.5 specifies maximum gain. The first of these values is equivalent to the gain for the left channel and the second value is equivalent to the gain for the right channel.

`siTwosComplementOnOff`
: Get or set the current state of the two's complement feature. This selector only applies to 8-bit data. (16-bit samples are always stored in two's complement format.) If on, the driver stores all samples in the application buffer as two's complement values (that is, –128 to 127). Otherwise, the driver stores the samples as offset binary values (that is, 0 to 255). The `infoData` parameter points to an integer, which is the current state of the two's complement feature (1 if two's complement output is desired, 0 otherwise).

`siUserInterruptProc`
: The Sound Input Manager sends this selector to specify the sound input interrupt routine that the sound input device driver should call. Your application should never issue this selector directly. The `infoData` parameter points to a procedure pointer, which is the address of the sound input interrupt routine.

`siVoxRecordInfo`
: Get or set the current VOX recording parameters. The `infoData` parameter points to two integers. The first integer indicates whether VOX recording is on or off (0 if off, 1 if on). The second integer indicates the VOX record trigger value. Trigger values range from 0 to 255 (0 is trigger immediately, 255 is trigger only on full volume).

`siVoxStopInfo`
: Get or set the current VOX stopping parameters. The `infoData` parameter points to three integers. The first integer indicates whether VOX stopping is on or off (0 if off, 1 if on). The second integer indicates the VOX stop trigger value. Trigger values range from 0 to 255 (255 is stop immediately, 0 is stop only on total silence). The third integer indicates how many milliseconds the trigger value must be continuously valid for recording to be stopped. Delay values range from 0 to 65,535.

Data Structures

This section describes the sound input parameter block.

Sound Input Parameter Blocks

The `SPBRecord` and `SPBRecordToFile` functions require a pointer to a sound input parameter block that defines characteristics of the recording. If you define a sound input completion routine or a sound input interrupt routine, your routine receives a pointer to a sound input parameter block. If you are using only the Sound Input Manager's high-level `SndRecord` and `SndRecordToFile` functions, the operation of sound input parameter blocks is transparent to your application. A sound input parameter block is defined by the `SPB` data type.

```
TYPE SPB =
RECORD
   inRefNum:            LongInt;    {reference number of input device}
   count:               LongInt;    {number of bytes to record}
   milliseconds:        LongInt;    {number of milliseconds to record}
   bufferLength:        LongInt;    {length of buffer to record into}
   bufferPtr:           Ptr;        {pointer to buffer to record into}
   completionRoutine:   ProcPtr;    {pointer to a completion routine}
   interruptRoutine:    ProcPtr;    {pointer to an interrupt routine}
   userLong:            LongInt;    {for application's use}
   error:               OSErr;      {error returned after recording}
   unused1:             LongInt;    {reserved}
END;
```

Field descriptions

`inRefNum` The reference number of the sound input device (as received from the `SPBOpenDevice` function) from which the recording is to occur.

`count` On input, the number of bytes to record. On output, the number of bytes actually recorded. If this field specifies a longer recording time than the `milliseconds` field, then the `milliseconds` field is ignored on input.

`milliseconds` On input, the number of milliseconds to record. On output, the number of milliseconds actually recorded. If this field specifies a longer recording time than the `count` field, then the `count` field is ignored on input.

`bufferLength` The length of the buffer into which recorded sound data is placed. The recording time specified by the `count` or `milliseconds` field is truncated to fit into this length, if necessary.

`bufferPtr` A pointer to the buffer into which recorded data is placed. If this field is `NIL`, then the `count`, `milliseconds`, and `bufferLength` fields are ignored and the recording will continue indefinitely until the `SPBStopRecording` function is called. However, the data is

not stored anywhere, so setting this field to `NIL` is useful only if you want to do something in a sound input interrupt routine but do not want to save the recorded sound.

`completionRoutine`
A pointer to a completion routine that is called when the recording terminates as a result of your calling the `SPBStopRecording` function or when the limit specified by the `count` or `milliseconds` field is reached. The completion routine executes only if `SPBRecord` is called asynchronously and therefore is called at interrupt time.

`interruptRoutine`
A pointer to a routine that is called by asynchronous recording devices when their internal buffers are full. You can define a sound input interrupt routine to modify uncompressed sound samples before they are placed into the buffer specified in the `bufferPtr` parameter. The interrupt routine executes only if `SPBRecord` is called asynchronously and therefore is called at interrupt time.

`userLong` A long integer available for the application's own use. You can use this field, for instance, to pass a handle to an application-defined structure to the completion routine or to the interrupt routine.

`error` On exit, the error that occurred during recording. This field contains a value greater than 0 while recording unless an error occurs, in which case it contains a value less than 0 that indicates an operating system error. Your application can poll this field to check on the status of an asynchronous recording. If recording terminates without an error, this field contains 0.

`unused1` Reserved for use by Apple. You should always initialize this field to 0.

Sound Input Manager Routines

This section describes the routines provided by the Sound Input Manager. You can use these routines to

- record sounds using the sound recording dialog box
- open and close sound input devices
- record sounds directly from sound input devices
- get information about sound input devices and change device settings
- construct sound resource and file headers
- register sound input devices with the Sound Input Manager
- convert recording times between millisecond and byte values
- obtain information about the version of the Sound Input Manager that is running

The section "Application-Defined Routines" on page 3-53 describes the format of sound input completion routines and sound input interrupt routines.

Recording Sounds

The Sound Input Manager provides two high-level sound input functions, `SndRecord` and `SndRecordToFile`, for recording sound. These input routines are analogous to the two Sound Manager functions `SndPlay` and `SndStartFilePlay`. By using these high-level routines, you can be assured that your application presents a user interface that is consistent with that displayed by other applications doing sound input. Both `SndRecord` and `SndRecordToFile` attempt to record sound data from the sound input hardware currently selected in the Sound In control panel.

SndRecord

You can use the `SndRecord` function to record sound resources into memory.

```
FUNCTION SndRecord (filterProc: ProcPtr; corner: Point;
                    quality: OSType; VAR sndHandle: Handle):
                    OSErr;
```

`filterProc`
: A pointer to an event filter function that determines how user actions in the sound recording dialog box are filtered (similar to the `filterProc` parameter specified in a call to the `ModalDialog` procedure). By specifying your own filter function, you can override or add to the default actions of the items in the dialog box. If `filterProc` isn't `NIL`, `SndRecord` filters events by calling the function that `filterProc` points to.

`corner`
: The horizontal and vertical coordinates of the upper-left corner of the sound recording dialog box (in global coordinates).

`quality`
: The desired quality of the recorded sound.

`sndHandle`
: On entry, a handle to some storage space or `NIL`. On exit, a handle to a valid sound resource (or unchanged, if the call did not execute successfully).

DESCRIPTION

The `SndRecord` function records sound into memory. The recorded data has the structure of a format 1 `'snd '` resource and can later be played using the `SndPlay` function or can be stored as a resource. `SndRecord` displays a sound recording dialog box and is always called synchronously. Controls in the dialog box allow the user to start, stop, pause, and resume sound recording, as well as to play back the recorded sound. The dialog box also lists the remaining recording time and the current microphone sound level.

The `quality` parameter defines the desired quality of the recorded sound. Currently, three values are recognized for the `quality` parameter:

```
CONST
   siBestQuality        = 'best';   {the best quality available}
   siBetterQuality      = 'betr';   {a quality better than good}
   siGoodQuality        = 'good';   {a good quality}
```

The precise meanings of these parameters are defined by the sound input device driver. For Apple-supplied drivers, this parameter determines whether the recorded sound is to be compressed, and if so, whether at a 6:1 or a 3:1 ratio. The quality `siBestQuality` does not compress the sound and provides the best quality output, but at the expense of increased memory use. The quality `siBetterQuality` is suitable for most nonvoice recording, and `siGoodQuality` is suitable for voice recording.

The `sndHandle` parameter is a handle to some storage space. If the handle is `NIL`, the Sound Input Manager allocates a handle of the largest amount of space that it can find in your application's heap and returns this handle in the `sndHandle` parameter. The Sound Input Manager resizes the handle when the user clicks the Save button in the sound recording dialog box. If the `sndHandle` parameter passed to `SndRecord` is not `NIL`, the Sound Input Manager simply stores the recorded data in the location specified by that handle.

SPECIAL CONSIDERATIONS

Because the `SndRecord` function moves memory, you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SndRecord` function are

Trap macro	Selector
`_SoundDispatch`	$08040014

RESULT CODES

`noErr`	0	No error
`userCanceledErr`	–128	User canceled the operation
`siBadSoundInDevice`	–221	Invalid sound input device
`siUnknownQuality`	–232	Unknown quality

SEE ALSO

See the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for a complete description of event filter functions.

SndRecordToFile

You can use `SndRecordToFile` to record sound data into a file.

```
FUNCTION SndRecordToFile (filterProc: ProcPtr; corner: Point;
                          quality: OSType;
                          fRefNum: Integer): OSErr;
```

`filterProc`
A pointer to a function that determines how user actions in the sound recording dialog box are filtered.

`corner` The horizontal and vertical coordinates of the upper-left corner of the sound recording dialog box (in global coordinates).

`quality` The desired quality of the recorded sound, as described on page 3-28.

`fRefNum` The file reference number of an open file to save the audio data in.

DESCRIPTION

The `SndRecordToFile` function works just like `SndRecord` except that it stores the sound input data into a file. The resulting file is in either AIFF or AIFF-C format and contains the information necessary to play the file by using the Sound Manager's `SndStartFilePlay` function. The `SndRecordToFile` function is always called synchronously.

Your application must open the file specified in the `fRefNum` parameter before calling the `SndRecordToFile` function. Your application must close the file sometime after calling `SndRecordToFile`.

SPECIAL CONSIDERATIONS

Because the `SndRecordToFile` function moves memory, you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SndRecordToFile` function are

Trap macro	Selector
`_SoundDispatch`	$07080014

RESULT CODES

`noErr`	0	No error
`userCanceledErr`	–128	User canceled the operation
`siBadSoundInDevice`	–221	Invalid sound input device
`siUnknownQuality`	–232	Unknown quality

Opening and Closing Sound Input Devices

You can use the `SPBOpenDevice` function to open the default sound input device that the user has selected in the Sound In control panel or to open a specific sound input device. You must open a device before you can record from it by using `SPBRecord`, but the Sound Input Manager's high-level routines automatically open the default sound input device. You can close a sound input device by calling the `SPBCloseDevice` function.

SPBOpenDevice

You can use the `SPBOpenDevice` function to open a sound input device.

```
FUNCTION SPBOpenDevice (deviceName: Str255; permission: Integer;
                        VAR inRefNum: LongInt): OSErr;
```

`deviceName`
The name of the sound input device to open, or the empty string if the default sound input device is to be opened.

`permission`
A flag that indicates whether subsequent operations with that device are to be read/write or read-only.

`inRefNum` On exit, if the function is successful, a device reference number for the open sound input device.

DESCRIPTION

The `SPBOpenDevice` function attempts to open a sound input device having the name indicated by the `deviceName` parameter. If `SPBOpenDevice` succeeds, it returns a device reference number in the `inRefNum` parameter. The `permission` parameter indicates whether subsequent operations with that device are to be read/write or read-only. If the device is not already in use, read/write permission is granted; otherwise, only read-only operations are allowed. To make any recording requests or to call the `SPBSetDeviceInfo` function, read/write permission must be available. Use these constants to request the appropriate permission:

```
CONST
   siReadPermission     = 0;     {open device for reading}
   siWritePermission    = 1;     {open device for reading/writing}
```

You can request that the current default sound input device be opened by passing either a zero-length string or a `NIL` string as the `deviceName` parameter. If only one sound input device is installed, that device is used. Generally you should open the default device unless you specifically want to use some other device. You can get a list of the available devices by calling the `SPBGetIndexedDevice` function.

SPECIAL CONSIDERATIONS

Because the `SPBOpenDevice` function allocates memory, you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBOpenDevice` function are

Trap macro	Selector
`_SoundDispatch`	$05180014

RESULT CODES

`noErr`	0	No error
`permErr`	–54	Device already open for writing
`siBadDeviceName`	–228	Invalid device name

SPBCloseDevice

You can use the `SPBCloseDevice` function to close a sound input device.

```
FUNCTION SPBCloseDevice (inRefNum: LongInt): OSErr;
```

`inRefNum` The device reference number of the sound input device to close.

DESCRIPTION

The `SPBCloseDevice` function closes a device that was previously opened by `SPBOpenDevice` and whose device reference number is specified in the `inRefNum` parameter.

SPECIAL CONSIDERATIONS

Because the `SPBCloseDevice` function moves or purges memory, you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBCloseDevice` function are

Trap macro	Selector
`_SoundDispatch`	$021C0014

RESULT CODES

noErr	0	No error
siBadRefNum	–229	Invalid reference number

Recording Sounds Directly From Sound Input Devices

The Sound Input Manager provides a number of routines that allow you to begin, pause, resume, and stop recording directly from a sound input device. These low-level routines do not display the sound recording dialog box to the user.

SPBRecord

You can use the `SPBRecord` function to record audio data into memory, either synchronously or asynchronously.

```
FUNCTION SPBRecord (inParamPtr: SPBPtr; asynchFlag: Boolean):
                    OSErr;
```

`inParamPtr`
A pointer to a sound input parameter block.

`asynchFlag`
A Boolean value that specifies whether the recording occurs asynchronously (`TRUE`) or synchronously (`FALSE`).

You specify values and receive return values in the sound input parameter block.

Parameter block

→	inRefNum	LongInt	A reference number of a sound input device.
↔	count	LongInt	The number of bytes of recording.
↔	milliseconds	LongInt	The number of milliseconds of recording.
→	bufferLength	LongInt	The length of the buffer beginning at `bufferPtr`.
→	bufferPtr	Ptr	A pointer to a buffer for sampled-sound data.
→	completionRoutine	ProcPtr	A pointer to a completion routine.
→	interruptRoutine	ProcPtr	A pointer to an interrupt routine.
→	userLong	LongInt	Free for application's use.
←	error	OSErr	The error value returned after recording.
→	unused1	LongInt	Reserved.

Field descriptions

`inRefNum` The device reference number of the sound input device, as obtained from the `SPBOpenDevice` function.

`count` On input, the number of bytes to record. If this field indicates a longer recording time than the `milliseconds` field, then the

milliseconds field is ignored. On output, this field indicates the number of bytes actually recorded.

`milliseconds` On input, the number of milliseconds to record. If this field indicates a longer recording time than the `count` field, then the `count` field is ignored. On output, this field indicates the number of milliseconds actually recorded.

`bufferLength` The number of bytes in the buffer specified by the `bufferPtr` parameter. If this buffer length is too small to contain the amount of sampled-sound data specified in the `count` and `milliseconds` fields, then recording time is truncated so that the sampled-sound data fits in the buffer.

`bufferPtr` A pointer to the buffer for the sampled-sound data, or `NIL` if you wish to record sampled-sound data without saving it. On exit, this buffer contains the sampled-sound data, which is interleaved for stereo sound on a sample basis (or on a packet basis if the data is compressed). This buffer contains only sampled-sound data, so if you need a sampled sound header, you should set that up in a buffer before calling `SPBRecord` and then record into the buffer following the sound header.

`completionRoutine`
A pointer to a completion routine. This routine is called when the recording terminates (either after you call the `SPBStopRecording` function or when the prescribed limit is reached). The completion routine is called only for asynchronous recording.

`interruptRoutine`
A pointer to an interrupt routine. The interrupt routine specified in the `interruptRoutine` field is called by asynchronous recording devices when their internal buffers are full.

`userLong` A long integer that your application can use to pass data to your application's completion or interrupt routines.

`error` On exit, a value greater than 0 while recording unless an error occurs, in which case it contains a value less than 0 that indicates an operating system error. Your application can poll this field to check on the status of an asynchronous recording. If recording terminates without an error, this field contains 0.

`unused1` Reserved. You should set this field to 0 before calling `SPBRecord`.

DESCRIPTION

The `SPBRecord` function starts recording into memory from a device specified in a sound input parameter block. The sound data recorded is stored in the buffer specified by the `bufferPtr` and `bufferLength` fields of the parameter block. Recording lasts the longer of the times specified by the `count` and `milliseconds` fields of the parameter block, or until the buffer is filled. Recording is asynchronous if the `asynchFlag` parameter is `TRUE` and the specified sound input device supports asynchronous recording.

If the `bufferPtr` field of the parameter block contains `NIL`, then the `count`, `milliseconds`, and `bufferLength` fields are ignored, and the recording continues indefinitely until you call the `SPBStopRecording` function. In this case, the audio data is not saved anywhere; this feature is useful only if you want to do something in your interrupt routine and do not want to save the audio data. However, if the recording is synchronous and `bufferPtr` is `NIL`, `SPBRecord` returns the result code `siNoBufferSpecified`.

The `SPBRecord` function returns the value that the `error` field of the parameter block contains when recording finishes.

SPECIAL CONSIDERATIONS

You can call the `SPBRecord` function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBRecord` function are

Trap macro	Selector
`_SoundDispatch`	$03200014

RESULT CODES

`noErr`	0	No error
`siNoSoundInHardware`	–220	No sound input hardware available
`siBadSoundInDevice`	–221	Invalid sound input device
`siNoBufferSpecified`	–222	No buffer specified
`siDeviceBusyErr`	–227	Sound input device is busy

SEE ALSO

For an example of the use of the `SPBRecord` function, see Listing 3-1.

SPBRecordToFile

You can use the `SPBRecordToFile` function to record audio data into a file, either synchronously or asynchronously.

```
FUNCTION SPBRecordToFile (fRefNum: Integer; inParamPtr: SPBPtr;
                          asynchFlag: Boolean): OSErr;
```

`fRefNum` The file reference number of an open file in which to place the recorded sound data.

`inParamPtr`
A pointer to a sound input parameter block.

`asynchFlag`
A Boolean value that specifies whether the recording occurs asynchronously (`TRUE`) or synchronously (`FALSE`).

Parameter block

→	`inRefNum`	`LongInt`	A reference number of a sound input device.
↔	`count`	`LongInt`	The number of bytes of recording.
↔	`milliseconds`	`LongInt`	The number of milliseconds of recording.
→	`completionRoutine`	`ProcPtr`	A pointer to a completion routine.
→	`interruptRoutine`	`ProcPtr`	Unused.
→	`userLong`	`LongInt`	Free for application's use.
←	`error`	`OSErr`	The error value returned after recording.
→	`unused1`	`LongInt`	Reserved.

Field descriptions

`inRefNum` The device reference number of the sound input device, as obtained from the `SPBOpenDevice` function.

`count` On input, the number of bytes to record. If this field indicates a longer recording time than the `milliseconds` field, then the `milliseconds` field is ignored. On output, the number of bytes actually recorded.

`milliseconds` On input, the number of milliseconds to record. If this field indicates a longer recording time than the `count` field, then the `count` field is ignored. On output, the number of milliseconds actually recorded.

`completionRoutine`
A pointer to a completion routine. This routine is called when the recording terminates (after you call the `SPBStopRecording` function, when the prescribed limit is reached, or after an error occurs). The completion routine is called only for asynchronous recording.

`interruptRoutine`
Unused. You should set this field to `NIL` before calling `SPBRecordToFile`.

`userLong` A long integer that your application can use to pass data to your application's completion or interrupt routines.

`error` On exit, the error that occurred during recording. This field contains the number 1 while recording unless an error occurs, in which case it contains a value less than 0 that indicates an operating system error. Your application can poll this field to check on the status of an asynchronous recording. If recording terminates without an error, this field contains 0.

`unused1` Reserved. You should set this field to 0 before calling the `SPBRecordToFile` function.

DESCRIPTION

The `SPBRecordToFile` function starts recording from the specified device into a file. The sound data recorded is simply stored in the file, so it is up to your application to insert whatever headers are needed to play the sound with the Sound Manager. Your application must open the file specified by the `fRefNum` parameter with write access before calling `SPBRecordToFile`, and it must eventually close that file.

The fields in the parameter block specified by the `inParamPtr` parameter are identical to the fields in the parameter block passed to the `SPBRecord` function, except that the `bufferLength` and `bufferPtr` fields are not used. The `interruptRoutine` field is ignored by `SPBRecordToFile` because `SPBRecordToFile` copies data returned by the sound input device driver to disk during the sound input interrupt routine, but you should initialize this field to `NIL`.

The `SPBRecordToFile` function writes samples to disk in the same format that they are read in from the sound input device. If compression is enabled, then the samples written to the file are compressed. Multiple channels of sound are interleaved on a sample basis (or, for compressed sound data, on a packet basis). When you are recording 8-bit audio data to an `AIFF` file, you must set the `siTwosComplementOnOff` flag to so that the data is stored on disk in the two's-complement format. If you don't store the data in this format, it sounds distorted when you play it back.

If any errors occur during the file writing process, recording is suspended. All File Manager errors are returned through the function's return value if the routine is called synchronously. If the routine is called asynchronously and the completion routine is not `NIL`, the completion routine is called and is passed a single parameter on the stack that points to the sound input parameter block; any errors are returned in the `error` field of the sound input parameter block.

The `SPBRecordToFile` function returns the value that the `error` field of the parameter block contains when recording finishes.

SPECIAL CONSIDERATIONS

Because the `SPBRecordToFile` function moves or purges memory, you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBRecordToFile` function are

Trap macro	Selector
`_SoundDispatch`	$04240014

RESULT CODES

noErr	0	No error
permErr	–54	Attempt to open locked file for writing
siNoSoundInHardware	–220	No sound input hardware available
siBadSoundInDevice	–221	Invalid sound input device
siHardDriveTooSlow	–224	Hard drive too slow to record

SPBPauseRecording

You can use the SPBPauseRecording function to pause recording from a sound input device.

```
FUNCTION SPBPauseRecording (inRefNum: LongInt): OSErr;
```

inRefNum The device reference number of the sound input device, as obtained from the SPBOpenDevice function.

DESCRIPTION

The SPBPauseRecording function pauses recording from the device specified by the inRefNum parameter. The recording must be asynchronous for this call to have any effect.

SPECIAL CONSIDERATIONS

You can call the SPBPauseRecording function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the SPBPauseRecording function are

Trap macro	Selector
_SoundDispatch	$02280014

RESULT CODES

noErr	0	No error
siBadSoundInDevice	–221	Invalid sound input device

SPBResumeRecording

You can use the `SPBResumeRecording` function to resume recording from a sound input device.

```
FUNCTION SPBResumeRecording (inRefNum: LongInt): OSErr;
```

`inRefNum` The device reference number of the sound input device, as obtained from the `SPBOpenDevice` function.

DESCRIPTION

The `SPBResumeRecording` function resumes recording from the device specified by the `inRefNum` parameter. Recording on that device must previously have been paused by a call to the `SPBPauseRecording` function for `SPBResumeRecording` to have any effect.

SPECIAL CONSIDERATIONS

You can call the `SPBResumeRecording` function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBResumeRecording` function are

Trap macro	Selector
`_SoundDispatch`	$022C0014

RESULT CODES

`noErr`	0	No error
`siBadSoundInDevice`	–221	Invalid sound input device

SPBStopRecording

You can use the `SPBStopRecording` function to end a recording from a sound input device.

```
FUNCTION SPBStopRecording (inRefNum: LongInt): OSErr;
```

`inRefNum` The device reference number of the sound input device, as obtained from the `SPBOpenDevice` function.

DESCRIPTION

The `SPBStopRecording` function stops recording from the device specified by the `inRefNum` parameter. The recording must be asynchronous for `SPBStopRecording` to have any effect. When you call `SPBStopRecording`, the sound input completion routine specified in the `completionRoutine` field of the sound input parameter block is called and the `error` field of that parameter block is set to `abortErr`. If you are writing a device driver, you will receive a `KillIO Status` call. See the section "Writing a Sound Input Device Driver" beginning on page 3-13 for more information.

SPECIAL CONSIDERATIONS

You can call the `SPBStopRecording` function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBStopRecording` function are

Trap macro	Selector
`_SoundDispatch`	$02300014

RESULT CODES

`noErr`	0	No error
`siBadSoundInDevice`	–221	Invalid sound input device

SPBGetRecordingStatus

You can use `SPBGetRecordingStatus` to obtain recording status information about a sound input device.

```
FUNCTION SPBGetRecordingStatus (inRefNum: LongInt;
                            VAR recordingStatus: Integer;
                            VAR meterLevel: Integer;
                            VAR totalSamplesToRecord: LongInt;
                            VAR numberOfSamplesRecorded: LongInt;
                            VAR totalMsecsToRecord: LongInt;
                            VAR numberOfMsecsRecorded: LongInt):
                            OSErr;
```

`inRefNum` The device reference number of the sound input device, as obtained from the `SPBOpenDevice` function.

`recordingStatus`
The status of the recording. While the input device is recording, this parameter is set to a number greater than 0. When a recording terminates without an error, this parameter is set to 0. When an error occurs during

recording or the recording has been terminated by a call to the `SPBStopRecording` function, this parameter is less than 0 and contains an error code.

`meterLevel`
The current input signal level. This level ranges from 0 to 255.

`totalSamplesToRecord`
The total number of samples to record, including those samples already recorded.

`numberOfSamplesRecorded`
The number of samples already recorded.

`totalMsecsToRecord`
The total duration of recording time, including recording time already elapsed.

`numberOfMsecsRecorded`
The amount of recording time that has elapsed.

DESCRIPTION

The `SPBGetRecordingStatus` function returns, in its second through seventh parameters, information about the recording on the device specified by the `inRefNum` parameter.

SPECIAL CONSIDERATIONS

You can call the `SPBGetRecordingStatus` function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBGetRecordingStatus` function are

Trap macro	Selector
`_SoundDispatch`	$0E340014

RESULT CODES

`noErr`	0	No error
`siBadSoundInDevice`	–221	Invalid sound input device

Manipulating Device Settings

You can use the two functions `SPBGetDeviceInfo` and `SPBSetDeviceInfo` to read and change the settings of a sound input device.

SPBGetDeviceInfo

You can use the `SPBGetDeviceInfo` function to get information about the settings of a sound input device.

```
FUNCTION SPBGetDeviceInfo (inRefNum: LongInt; infoType: OSType;
                           infoData: Ptr): OSErr;
```

`inRefNum` The device reference number of the sound input device, as obtained from the `SPBOpenDevice` function.

`infoType` A sound input device information selector that specifies the type of information you need.

`infoData` A pointer to a buffer in which information should be returned. This buffer must be large enough for the type of information specified in the `infoType` parameter.

DESCRIPTION

The `SPBGetDeviceInfo` function returns information about the sound input device specified by the `inRefNum` parameter. The type of information you want is specified in the `infoType` parameter. The available sound input device information selectors are listed in "Sound Input Device Information Selectors" beginning on page 3-18. The information is copied into the buffer specified by the `infoData` parameter.

SPECIAL CONSIDERATIONS

Because the `SPBGetDeviceInfo` function might move memory, you should not call it at interrupt time. Check the selector description of the selector you want to use to see if it moves memory before calling the `SPBGetDeviceInfo` function. Most of the selectors do not move memory and are therefore safe to use at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBGetDeviceInfo` function are

Trap macro	Selector
`_SoundDispatch`	$06380014

RESULT CODES

`noErr`	0	No error
`siBadSoundInDevice`	–221	Invalid sound input device
`siUnknownInfoType`	–231	Unknown type of information

SEE ALSO

Listing 3-2 on page 3-12 shows an example that uses the `SPBGetDeviceInfo` function to get the name of a sound input device driver.

SPBSetDeviceInfo

You can use the `SPBSetDeviceInfo` function to set information in a sound input device.

```
FUNCTION SPBSetDeviceInfo (inRefNum: LongInt; infoType: OSType;
                           infoData: Ptr): OSErr;
```

`inRefNum` The device reference number of the sound input device, as obtained from the `SPBOpenDevice` function.

`infoType` A sound input device information selector that specifies the type of information you need.

`infoData` A pointer to a buffer. This buffer can contain information on entry, and information might be returned on exit. This buffer must be large enough for the type of information specified in the `infoType` parameter, and the data in the buffer must be set to appropriate values if information needs to be passed in to the `SPBSetDeviceInfo` function.

DESCRIPTION

The `SPBSetDeviceInfo` function sets information about the sound input device specified by the `inRefNum` parameter, based on the data in the buffer specified by the `infoData` parameter.

The type of setting you wish to change is specified in the `infoType` parameter. The sound input device information selectors are listed in "Sound Input Device Information Selectors" beginning on page 3-18.

SPECIAL CONSIDERATIONS

Because the `SPBSetDeviceInfo` function might move memory, you should not call it at interrupt time. Check the selector description of the selector you want to use to see if it moves memory before calling the `SPBGetDeviceInfo` function. Most of the selectors do not move memory and are therefore safe to use at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBSetDeviceInfo` function are

Trap macro	Selector
`_SoundDispatch`	$063C0014

RESULT CODES

`noErr`	0	No error
`permErr`	–54	Attempt to open locked file for writing
`siBadSoundInDevice`	–221	Invalid sound input device
`siDeviceBusyErr`	–227	Sound input device is busy
`siUnknownInfoType`	–231	Unknown type of information

Constructing Sound Resource and File Headers

The Sound Input Manager provides two functions, `SetupSndHeader` and `SetupAIFFHeader`, to help you set up headers for sound resources and sound files.

SetupSndHeader

You can use the `SetupSndHeader` function to construct a sound resource containing sampled sound that can be passed to the `SndPlay` function.

```
FUNCTION SetupSndHeader (sndHandle: Handle;
                         numChannels: Integer;
                         sampleRate: Fixed;
                         sampleSize: Integer;
                         compressionType: OSType;
                         baseFrequency: Integer;
                         numBytes: LongInt;
                         VAR headerLen: Integer): OSErr;
```

`sndHandle` A handle to a block of memory that is at least large enough to store the sound resource header information. The handle is not resized in any way upon successful completion of `SetupSndHeader`. The `SetupSndHeader` function simply fills the relocatable block specified by this parameter with the header information needed for a format 1 `'snd '` resource, including the sound resource header, the list of sound commands, and a sampled sound header. It is your application's responsibility to append the desired sampled-sound data.

`numChannels`
The number of channels for the sound; one channel is equivalent to monaural sound and two channels are equivalent to stereo sound.

`sampleRate`
The rate at which the sound was recorded. The sample rate is declared as a `Fixed` data type. In order to accommodate sample rates greater than 32 kHz, the most significant bit is not treated as a sign bit; instead, that bit is interpreted as having the value 32,768.

`sampleSize`
The sample size for the original sound (that is, bits per sample).

`compressionType`
The compression type for the sound (`'NONE'`, `'MAC3'`, `'MAC6'`, or other third-party types).

`baseFrequency`
The base frequency for the sound, expressed as a MIDI note value.

`numBytes` The number of bytes of audio data that are to be stored in the handle. (This value is not necessarily the same as the number of samples in the sound.)

`headerLen` On exit, the size (in bytes) of the `'snd '` resource header that is created. In no case will this length exceed 100 bytes. This field allows you to put the audio data right after the header in the relocatable block specified by the `sndHandle` parameter. The value returned depends on the type of sound header created.

DESCRIPTION

The `SetupSndHeader` function creates a format 1 `'snd '` resource for a sampled sound. The resource contains a sound resource header that links the sound to the sampled synthesizer, a single sound command (a `bufferCmd` command to play the accompanying data), and a sampled sound header. You can use `SetupSndHeader` to construct a sampled sound header that can be passed to the Sound Manager's `SndPlay` function or stored as an `'snd '` resource. After calling the `SetupSndHeader` function, your application should place the sampled-sound data directly after the sampled sound header so that, in essence, the sampled sound header's final field contains the sound data.

The sampled sound is in one of three formats depending on several of the parameters passed. Table 3-1 shows how `SetupSndHeader` determines what kind of sound header to create.

Table 3-1 The sampled sound header format used by `SetupSndHeader`

compressionType	numChannels	sampleSize	Sampled sound header format
`'NONE'`	1	8	`SoundHeader`
`'NONE'`	1	16	`ExtSoundHeader`
`'NONE'`	2	any	`ExtSoundHeader`
not `'NONE'`	any	any	`CmpSoundHeader`

A good way to use this function is to create a handle in which you want to store a sampled sound, then call `SetupSndHeader` with the `numBytes` parameter set to `0` to see how much room the header for that sound will occupy and hence where to append the audio data. Then record the data into the handle and call `SetupSndHeader` again with `numBytes` set to the correct amount of sound data recorded. The handle filled out in this way can be passed to `SndPlay` to play the sound.

SPECIAL CONSIDERATIONS

You cannot call the `SetupSndHeader` function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SetupSndHeader` function are

Trap macro	Selector
`_SoundDispatch`	$0D480014

RESULT CODES

`noErr`	0	No error
`siInvalidCompression`	–223	Invalid compression type

SEE ALSO

For an example that uses the `SetupSndHeader` function to set up a sound header before recording, see Listing 3-1 on page 3-7.

SetupAIFFHeader

You can use the `SetupAIFFHeader` function to set up a file that can subsequently be played by `SndStartFilePlay`.

```
FUNCTION SetupAIFFHeader (fRefNum: Integer;
                          numChannels: Integer;
                          sampleRate: Fixed;
                          sampleSize: Integer;
                          compressionType: OSType;
                          numBytes: LongInt;
                          numFrames: LongInt): OSErr;
```

`fRefNum` A file reference number of a file that is open for writing.

`numChannels`
The number of channels for the sound; one channel is equivalent to monaural sound and two channels are equivalent to stereo sound.

`sampleRate`
The rate at which the sound was recorded. The sample rate is declared as a `Fixed` data type. In order to accommodate sample rates greater than 32 kHz, the most significant bit is not treated as a sign bit; instead, that bit is interpreted as having the value 32,768.

`sampleSize`
The sample size for the original sound (that is, bits per sample).

`compressionType`
: The compression type for the sound (`'NONE'`, `'MAC3'`, `'MAC6'`, or other third-party types).

`numBytes`
: The number of bytes of audio data that are to be stored in the Common Chunk of the AIFF or AIFF-C file.

`numFrames`
: The number of sample frames for the sample sound. If you are using a compression type defined by Apple, you can pass 0 in this field and the appropriate value for this field will be computed automatically.

DESCRIPTION

The `SetupAIFFHeader` function creates an AIFF or AIFF-C file header, depending on the parameters passed to it:

- Uncompressed sounds of any type are stored in AIFF format (that is, the `compressionType` parameter is `'NONE'`).
- Compressed sounds of any type are stored in AIFF-C format (that is, the `compressionType` parameter is different from `'NONE'`).

Note
The `SetupAIFFHeader` function might format a sound file as an AIFF file even if the File Manager file type of a file is `'AIFC'`. The Sound Manager will still play such files correctly. ◆

The AIFF header information is written starting at the current file position of the file specified by the `fRefNum` parameter, and the file position is left at the end of the header upon completion. The `SetupAIFFHeader` function creates a Form Chunk, a Format Version Chunk, a Common Chunk, and a Sound Data chunk, but it does not put any sound data at the end of the Sound Data Chunk.

A good way to use this routine is to create a file that you want to store a sound in, then call `SetupAIFFHeader` with `numBytes` set to 0 to position the file to be ready to write the audio data. Then record the data to the file, set the file position to the beginning of the file, and call `SetupAIFFHeader` again with `numBytes` set to the correct amount of sound data recorded. The file created in this way can be passed to the `SndStartFilePlay` function to play the sound.

SPECIAL CONSIDERATIONS

If recording produces an odd number of bytes of sound data, you must add a pad byte to make the total number of bytes even.

Because the `SetupAIFFHeader` function moves memory, you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SetupAIFFHeader` function are

Trap macro	Selector
`_SoundDispatch`	$0B4C0014

RESULT CODES

`noErr`	0	No error
`siInvalidCompression`	–223	Invalid compression type

Registering Sound Input Devices

Sound input device drivers must call the `SPBSignInDevice` function to register with the Sound Input Manager before they can use its sound input services. You might call this routine at system startup time from within an extension to install a sound input device driver. Your application can generate a list of registered sound input devices by using the `SPBGetIndexedDevice` function. You can cancel the registration of your driver, thus removing it from the Sound control panel and making it inaccessible, by calling the `SPBSignOutDevice` function.

SPBSignInDevice

You can register a sound input device by calling the `SPBSignInDevice` function.

```
FUNCTION SPBSignInDevice (deviceRefNum: Integer;
                          deviceName: Str255): OSErr;
```

`deviceRefNum`
The device driver reference number of the sound input device to register with the Sound Input Manager.

`deviceName`
The device's name as it is to appear to the user in the Sound In control panel (which is not the name of the driver used by the Device Manager).

DESCRIPTION

The `SPBSignInDevice` function registers with the Sound Input Manager the device whose driver reference number is `deviceRefNum`.

The `deviceName` parameter specifies this device's name as it is to appear to the user in the Sound In control panel (which is not the name of the driver itself). Accordingly, the name should be as descriptive as possible. You should call `SPBSignInDevice` after you have already opened your driver by calling normal Device Manager routines.

SPECIAL CONSIDERATIONS

Because the `SPBSignInDevice` function moves or purges memory, you should not call it at interrupt time. You can, however, call it at system startup time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBSignInDevice` function are

Trap macro	Selector
`_SoundDispatch`	$030C0014

RESULT CODES

`noErr`	0	No error
`siBadSoundInDevice`	–221	Invalid sound input device

SPBGetIndexedDevice

You can use the `SPBGetIndexedDevice` function to help generate a list of sound input devices.

```
FUNCTION SPBGetIndexedDevice (count: Integer;
                              VAR deviceName: Str255;
                              VAR deviceIconHandle: Handle):
                              OSErr;
```

`count` The index number of the sound input device you wish to obtain information about.

`deviceName`
On exit, the name of the sound input device specified by the `count` parameter.

`deviceIconHandle`
On exit, a handle to the icon of the sound input device specified by the `count` parameter. The memory for this icon is allocated automatically, but your application must dispose of it.

DESCRIPTION

The `SPBGetIndexedDevice` function returns the name and icon of the device whose index is specified in the `count` parameter. Your application can create a list of sound input devices by calling this function with a count starting at 1 and incrementing it by 1 until the function returns `siBadSoundInDevice`.

Because the Sound In control panel allows the user to select a sound input device, most applications should not use this function. Your application might need to use this function if it allows the user to record from more than one sound input device at once.

SPECIAL CONSIDERATIONS

Because the `SPBGetIndexedDevice` function allocates memory, you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBGetIndexedDevice` function are

Trap macro	Selector
`_SoundDispatch`	$05140014

RESULT CODES

`noErr`	0	No error
`siBadSoundInDevice`	–221	Invalid sound input device

SPBSignOutDevice

You can use the `SPBSignOutDevice` function to cancel the registration of a device you have previously registered with the `SPBSignInDevice` function.

```
FUNCTION SPBSignOutDevice (deviceRefNum: Integer): OSErr;
```

`deviceRefNum`
The driver reference number of the device you wish to sign out.

DESCRIPTION

The `SPBSignOutDevice` function cancels the registration of the device whose driver reference number is `deviceRefNum`; the device is unregistered from the Sound Input Manager's list of available sound input devices and no longer appears in the Sound In control panel.

Ordinarily, you should not need to use the `SPBSignOutDevice` function. You might use it if your device driver detects that a sound input device is not functioning correctly or has been disconnected.

SPECIAL CONSIDERATIONS

Because the `SPBSignOutDevice` function moves or purges memory, you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBSignOutDevice` function are

Trap macro	Selector
`_SoundDispatch`	$01100014

RESULT CODES

`noErr`	0	No error
`siBadSoundInDevice`	–221	Invalid sound input device
`siDeviceBusyErr`	–227	Sound input device is busy

Converting Between Milliseconds and Bytes

The Sound Input Manager provides two routines that allow you to convert between millisecond and byte recording values.

SPBMilliSecondsToBytes

You can use the `SPBMilliSecondsToBytes` function to determine how many bytes a recording of a certain duration will use.

```
FUNCTION SPBMilliSecondsToBytes (inRefNum: LongInt;
                                 VAR milliseconds: LongInt): OSErr;
```

`inRefNum` The device reference number of the sound input device, as obtained from the `SPBOpenDevice` function.

`milliseconds`
On entry, the duration of the recording in milliseconds. On exit, the number of bytes that sampled-sound data would occupy for a recording of the specified duration on the device specified by the `inRefNum` parameter.

DESCRIPTION

The `SPBMilliSecondsToBytes` function reports how many bytes are required to store a recording of duration `milliseconds`, given the input device's current sample rate, sample size, number of channels, and compression factor.

SPECIAL CONSIDERATIONS

You can call the `SPBMilliSecondsToBytes` function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBMilliSecondsToBytes` function are

Trap macro	Selector
`_SoundDispatch`	$04400014

RESULT CODES

`noErr`	0	No error
`siBadSoundInDevice`	–221	Invalid sound input device

SPBBytesToMilliSeconds

You can use the `SPBBytesToMilliSeconds` function to determine the maximum duration of a recording that can fit in a buffer of a certain size.

```
FUNCTION SPBBytesToMilliSeconds (inRefNum: LongInt;
                                 VAR byteCount: LongInt): OSErr;
```

`inRefNum` The device reference number of the sound input device, as obtained from the `SPBOpenDevice` function.

`byteCount` On entry, a value in bytes. On exit, the number of milliseconds of recording on the device specified by the `inRefNum` parameter that would be necessary to fill a buffer of such a size.

DESCRIPTION

The `SPBBytesToMilliSeconds` function reports how many milliseconds of audio data can be recorded in a buffer that is `byteCount` bytes long, given the input device's current sample rate, sample size, number of channels, and compression factor.

SPECIAL CONSIDERATIONS

You can call the `SPBBytesToMilliSeconds` function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBBytesToMilliSeconds` function are

Trap macro	Selector
`_SoundDispatch`	$04440014

RESULT CODES

`noErr`	0	No error
`siBadSoundInDevice`	–221	Invalid sound input device

Obtaining Information

The `SPBVersion` function allows you to determine the version of the Sound Input Manager.

SPBVersion

You can use the `SPBVersion` function to determine the version of the sound input tools available on a machine.

```
FUNCTION SPBVersion: NumVersion;
```

DESCRIPTION

The `SPBVersion` function returns a version number that contains the same information as in the first 4 bytes of a `'vers'` resource or a `NumVersion` data type. For a description of the version record, see the chapter "Sound Manager" in this book.

SPECIAL CONSIDERATIONS

You can call the `SPBVersion` function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SPBVersion` function are

Trap macro	Selector
`_SoundDispatch`	$00000014

SEE ALSO

For a complete discussion of `'vers'` resources, see the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials*.

Application-Defined Routines

This section describes the routines that your application or device driver might need to define. Your application can define a sound input completion routine to perform an action when recording finishes, and your application can define a sound input interrupt routine to manipulate sound data during recording.

Sound Input Completion Routines

You can specify a sound input completion routine in the `completionRoutine` field of a sound input parameter block that your application uses to initiate asynchronous recording directly from a device.

MySICompletionRoutine

A sound input completion routine has the following syntax:

```
PROCEDURE MySICompletionRoutine (inParamPtr: SPBPtr);
```

`inParamPtr`
A pointer to the sound input parameter block that was used to initiate an asynchronous recording.

DESCRIPTION

The Sound Input Manager executes your sound input completion routine after recording terminates either because your application has called the `SPBStopRecording` function or because the prescribed limit is reached. The completion routine is called only for asynchronous recording.

A common use of a sound input completion routine is to set a global variable that alerts the application that it should dispose of a sound input parameter block that it had allocated for an asynchronous sound recording.

SPECIAL CONSIDERATIONS

Because a sound input completion routine is executed at interrupt time, it should not allocate, move, or purge memory (either directly or indirectly) and should not depend on the validity of handles to unlocked blocks.

If your sound input completion routine accesses your application's global variables, it must ensure that the A5 register contains the address of the boundary between the application global variables and the application parameters. Your application can pass the value of the A5 register to the sound input completion routine in the `userLong` field of the sound input parameter block. For more information on ensuring the validity of the A5 register, see the chapter "Memory Management Utilities" in *Inside Macintosh: Memory*.

Your sound input completion routine can determine whether an error occurred during recording by examining the `error` field of the sound input parameter block specified by `inParamPtr`. Your sound input completion routine can change the value of that field to alert the application that some other error has occurred.

ASSEMBLY-LANGUAGE INFORMATION

Because a sound input completion routine is called at interrupt time, it must preserve all registers other than A0–A1 and D0–D2.

RESULT CODES

`noErr`	0	No error
`abortErr`	–27	Asynchronous recording was cancelled
`siNoSoundInHardware`	–220	No sound input hardware available
`siBadSoundInDevice`	–221	Invalid sound input device
`siNoBufferSpecified`	–222	No buffer specified
`siDeviceBusyErr`	–227	Sound input device is busy

Sound Input Interrupt Routines

You can specify a sound input interrupt routine in the `interruptRoutine` field of the sound input parameter block that your application uses to initiate asynchronous recording directly from a device. Because the `SPBRecordToFile` function uses sound input interrupt routines to enable it to record sound data to disk during recording, you can use sound input interrupt routines only with the `SPBRecord` function.

MySIInterruptRoutine

A sound input interrupt routine has the following syntax:

```
PROCEDURE MySIInterruptRoutine;
```

DESCRIPTION

A sound input device driver executes the sound input interrupt routine associated with an asynchronous sound recording whenever the driver's internal buffers are full. The internal buffers contain raw samples taken directly from the input device. The interrupt routine can thus modify the samples in the buffer in any way it requires. After your sound input interrupt routine finishes processing the data, the sound input device driver compresses the data (if compression is enabled) and copies the data into your application's buffer.

SPECIAL CONSIDERATIONS

If your sound input interrupt routine accesses your application's global variables, it must ensure that the A5 register contains the address of the boundary between the application global variables and the application parameters. Your application can pass the value of the A5 register to the sound input interrupt routine in the `userLong` field of the sound input parameter block. For more information on ensuring the validity of the A5 register, see the chapter "Memory Management Utilities" in *Inside Macintosh: Memory.*

ASSEMBLY-LANGUAGE INFORMATION

Sound input interrupt routines are sometimes written in assembly language to maximize real-time performance in recording sound. On entry, registers are set up as follows:

Registers on entry

A0	Address of the sound parameter block passed to `SPBRecord`
A1	Address of the start of the sample buffer
D0	Peak amplitude for sample buffer if metering is on
D1	Size of the sample buffer in bytes

If you write a sound input interrupt routine in a high-level language like Pascal or C, you might need to write inline code to copy variables from the registers into local variables that your application defines.

Because a sound input interrupt routine is called at interrupt time, it must preserve all registers.

Summary of the Sound Input Manager

Pascal Summary

Constants

```
CONST
   gestaltSoundAttr          = 'snd ';{sound attributes selector}

   {Gestalt response bit flags related to sound input}
   gestaltSoundIOMgrPresent   = 3;     {sound input routines available}
   gestaltBuiltInSoundInput   = 4;     {built-in input hw available}
   gestaltHasSoundInputDevice = 5;     {sound input device available}
   gestaltPlayAndRecord       = 6;     {built-in hw can play while recording}
   gestalt16BitSoundIO        = 7;     {built-in hw can handle 16-bit data}
   gestaltStereoInput         = 8;     {built-in hw can record stereo sounds}
   gestaltLineLevelInput      = 9;     {built-in input hw needs line level}

   {available information selectors for sound input device drivers}
   siActiveChannels        = 'chac';   {channels active}
   siActiveLevels          = 'lmac';   {levels active}
   siAGCOnOff              = 'agc ';   {automatic gain control state}
   siAsync                 = 'asyn';   {asynchronous capability}
   siChannelAvailable      = 'chav';   {number of channels available}
   siCompressionAvailable  = 'cmav';   {compression types available}
   siCompressionFactor     = 'cmfa';   {current compression factor}
   siCompressionHeader     = 'cmhd';   {return compression header}
   siCompressionNames      = 'cnam';   {return compression type names}
   siCompressionType       = 'comp';   {current compression type}
   siContinuous            = 'cont';   {continuous recording}
   siDeviceBufferInfo      = 'dbin';   {size of interrupt buffer}
   siDeviceConnected       = 'dcon';   {input device connection status}
   siDeviceIcon            = 'icon';   {input device icon}
   siDeviceName            = 'name';   {input device name}
   siInputGain             = 'gain';   {input gain level}
   siInputSource           = 'sour';   {input source selector}
   siInputSourceNames      = 'snam';   {input source names}
   siLevelMeterOnOff       = 'lmet';   {level meter state}
   siNumberChannels        = 'chan';   {current number of channels}
```

```
   siOptionsDialog         = 'optd';   {display options dialog box}
   siPlayThruOnOff         = 'plth';   {play-through state}
   siRecordingQuality      = 'qual';   {recording quality}
   siSampleRate            = 'srat';   {current sample rate}
   siSampleRateAvailable   = 'srav';   {sample rates available}
   siSampleSize            = 'ssiz';   {current sample size}
   siSampleSizeAvailable   = 'ssav';   {sample sizes available}
   siStereoInputGain       = 'sgai';   {stereo input gain level}
   siTwosComplementOnOff   = 'twos';   {two's complement state}
   siVoxRecordInfo         = 'voxr';   {VOX record parameters}
   siVoxStopInfo           = 'voxs';   {VOX stop parameters}

   {internal information selectors for sound input device drivers}
   siCloseDriver           = 'clos';   {release driver}
   siInitializeDriver      = 'init';   {initialize driver}
   siPauseRecording        = 'paus';   {pause recording}
   siUserInterruptProc     = 'user';   {set sound input interrupt routine}

   {sound-recording qualities}
   siBestQuality           = 'best';   {the best quality available}
   siBetterQuality         = 'betr';   {a quality better than good}
   siGoodQuality           = 'good';   {a good quality}

   {sound input device permissions}
   siReadPermission        = 0;        {open device for reading}
   siWritePermission       = 1;        {open device for reading/writing}

   {device-connection states}
   siDeviceIsConnected     = 1;        {device is connected and ready}
   siDeviceNotConnected    = 0;        {device is not connected}
   siDontKnowIfConnected   = -1;       {can't tell if device is connected}
```

Data Types

Sound Input Parameter Block

```
TYPE SPB =
RECORD
   inRefNum:            LongInt;    {reference number of input device}
   count:               LongInt;    {number of bytes to record}
   milliseconds:        LongInt;    {number of milliseconds to record}
   bufferLength:        LongInt;    {length of buffer to record into}
   bufferPtr:           Ptr;        {pointer to buffer to record into}
   completionRoutine:   ProcPtr;    {pointer to a completion routine}
```

```
   interruptRoutine:    ProcPtr;    {pointer to an interrupt routine}
   userLong:            LongInt;    {for application's use}
   error:               OSErr;      {error returned after recording}
   unused1:             LongInt;    {reserved}
END;
SPBPtr = ^SPB;
```

Sound Input Manager Routines

Recording Sounds

```
FUNCTION SndRecord          (filterProc: ProcPtr; corner: Point;
                             quality: OSType; VAR sndHandle: Handle): OSErr;
FUNCTION SndRecordToFile    (filterProc: ProcPtr; corner: Point;
                             quality: OSType; fRefNum: Integer): OSErr;
```

Opening and Closing Sound Input Devices

```
FUNCTION SPBOpenDevice      (deviceName: Str255; permission: Integer;
                             VAR inRefNum: LongInt): OSErr;
FUNCTION SPBCloseDevice     (inRefNum: LongInt): OSErr;
```

Recording Sounds Directly From Sound Input Devices

```
FUNCTION SPBRecord          (inParamPtr: SPBPtr; asynchFlag: Boolean):
                             OSErr;
FUNCTION SPBRecordToFile    (fRefNum: Integer; inParamPtr: SPBPtr;
                             asynchFlag: Boolean): OSErr;
FUNCTION SPBPauseRecording  (inRefNum: LongInt): OSErr;
FUNCTION SPBResumeRecording
                            (inRefNum: LongInt): OSErr;
FUNCTION SPBStopRecording   (inRefNum: LongInt): OSErr;
FUNCTION SPBGetRecordingStatus
                            (inRefNum: LongInt;
                             VAR recordingStatus: Integer;
                             VAR meterLevel: Integer;
                             VAR totalSamplesToRecord: LongInt;
                             VAR numberOfSamplesRecorded: LongInt;
                             VAR totalMsecsToRecord: LongInt;
                             VAR numberOfMsecsRecorded: LongInt): OSErr;
```

Manipulating Device Settings

```
FUNCTION SPBGetDeviceInfo   (inRefNum: LongInt; infoType: OSType;
                             infoData: Ptr): OSErr;
```

```
FUNCTION SPBSetDeviceInfo   (inRefNum: LongInt; infoType: OSType;
                             infoData: Ptr): OSErr;
```

Constructing Sound Resource and File Headers

```
FUNCTION SetupSndHeader     (sndHandle: Handle; numChannels: Integer;
                             sampleRate: Fixed; sampleSize: Integer;
                             compressionType: OSType;
                             baseFrequency: Integer; numBytes: LongInt;
                             VAR headerLen: Integer): OSErr;
FUNCTION SetupAIFFHeader    (fRefNum: Integer; numChannels: Integer;
                             sampleRate: Fixed; sampleSize: Integer;
                             compressionType: OSType; numBytes: LongInt;
                             numFrames: LongInt): OSErr;
```

Registering Sound Input Devices

```
FUNCTION SPBSignInDevice    (deviceRefNum: Integer; deviceName: Str255):
                             OSErr;
FUNCTION SPBGetIndexedDevice
                            (count: Integer; VAR deviceName: Str255;
                             VAR deviceIconHandle: Handle): OSErr;
FUNCTION SPBSignOutDevice   (deviceRefNum: Integer): OSErr;
```

Converting Between Milliseconds and Bytes

```
FUNCTION SPBMilliSecondsToBytes
                            (inRefNum: LongInt; VAR milliseconds: LongInt):
                             OSErr;
FUNCTION SPBBytesToMilliSeconds
                            (inRefNum: LongInt; VAR byteCount: LongInt):
                             OSErr;
```

Obtaining Information

```
FUNCTION SPBVersion         : NumVersion;
```

Application-Defined Routines

```
PROCEDURE MySICompletionRoutine
                            (inParamPtr: SPBPtr);
PROCEDURE MySIInterruptRoutine;
```

C Summary

Constants

```
#define gestaltSoundAttr    'snd '  /*sound attributes selector*/

enum {
   /*Gestalt response bit flags related to sound input*/
   gestaltSoundIOMgrPresent   = 3,  /*sound input routines available*/
   gestaltBuiltInSoundInput   = 4,  /*built-in input hw available*/
   gestaltHasSoundInputDevice = 5,  /*sound input device available*/
   gestaltPlayAndRecord       = 6,  /*built-in hw can play while recording*/
   gestalt16BitSoundIO        = 7,  /*built-in hw can handle 16-bit data*/
   gestaltStereoInput         = 8,  /*built-in hw can record stereo sounds*/
   gestaltLineLevelInput      = 9   /*built-in input hw needs line level*/
};

/*available information selectors for sound input device drivers*/
#define siActiveChannels      'chac'   /*channels active*/
#define siActiveLevels        'lmac'   /*levels active*/
#define siAGCOnOff            'agc '   /*automatic gain control state*/
#define siAsync               'asyn'   /*asynchronous capability*/
#define siChannelAvailable    'chav'   /*number of channels available*/
#define siCompressionAvailable'cmav'   /*compression types available*/
#define siCompressionFactor   'cmfa'   /*current compression factor*/
#define siCompressionHeader   'cmhd'   /*return compression header*/
#define siCompressionNames    'cnam'   /*return compression type names*/
#define siCompressionType     'comp'   /*current compression type*/
#define siContinuous          'cont'   /*continuous recording*/
#define siDeviceBufferInfo    'dbin'   /*size of interrupt buffer*/
#define siDeviceConnected     'dcon'   /*input device connection status*/
#define siDeviceIcon          'icon'   /*input device icon*/
#define siDeviceName          'name'   /*input device name*/
#define siInputGain           'gain'   /*input gain level*/
#define siInputSource         'sour'   /*input source selector*/
#define siInputSourceNames    'snam'   /*input source names*/
#define siLevelMeterOnOff     'lmet'   /*level meter state*/
#define siNumberChannels      'chan'   /*current number of channels*/
#define siOptionsDialog       'optd'   /*display options dialog box*/
#define siPlayThruOnOff       'plth'   /*play-through state*/
#define siRecordingQuality    'qual'   /*recording quality*/
#define siSampleRate          'srat'   /*current sample rate*/
#define siSampleRateAvailable 'srav'   /*sample rates available*/
```

```
#define siSampleSize          'ssiz'   /*current sample size*/
#define siSampleSizeAvailable 'ssav'   /*sample sizes available*/
#define siStereoInputGain     'sgai'   /*stereo input gain level*/
#define siTwosComplementOnOff 'twos'   /*two's complement state*/
#define siVoxRecordInfo       'voxr'   /*VOX record parameters*/
#define siVoxStopInfo         'voxs'   /*VOX stop parameters*/

/*internal information selectors for sound input device drivers*/
#define siCloseDriver         'clos'   /*release driver*/
#define siInitializeDriver    'init'   /*initialize driver*/
#define siPauseRecording      'paus'   /*pause recording*/
#define siUserInterruptProc   'user'   /*set sound input interrupt routine*/

/*sound-recording qualities*/
#define siBestQuality         'best'   /*the best quality available*/
#define siBetterQuality       'betr'   /*a quality better than good*/
#define siGoodQuality         'good'   /*a good quality*/

/*sound input device permissions*/
enum {
   siReadPermission           = 0,     /*open device for reading*/
   siWritePermission          = 1      /*open device for reading/writing*/
};

/*device-connection states*/
enum {
   siDeviceIsConnected        = 1,     /*device is connected and ready*/
   siDeviceNotConnected       = 0,     /*device is not connected*/
   siDontKnowIfConnected      = -1     /*can't tell if device is connected*/
};
```

Data Types

Sound Input Parameter Block

```
struct SPB {
   long                 inRefNum;      /*reference number of input device*/
   unsigned long        count;         /*number of bytes to record*/
   unsigned long        milliseconds;  /*number of milliseconds to record*/
   unsigned long        bufferLength;  /*length of buffer to record into*/
   Ptr                  bufferPtr;     /*pointer to buffer to record into*/
   ProcPtr              completionRoutine;
                                       /*pointer to a completion routine*/
   ProcPtr              interruptRoutine;
```

```
                                        /*pointer to an interrupt routine*/
   long                 userLong;       /*for application's use*/
   OSErr                error;          /*error returned after recording*/
   long                 unused1;        /*reserved*/
};
typedef struct SPB SPB;
typedef SPB *SPBPtr;
```

Sound Input Manager Routines

Recording Sounds

```
pascal OSErr SndRecord      (ModalFilterProcPtr filterProc, Point corner,
                             OSType quality, Handle *sndHandle);
pascal OSErr SndRecordToFile
                            (ModalFilterProcPtr filterProc, Point corner,
                             OSType quality, short fRefNum);
```

Opening and Closing Sound Input Devices

```
pascal OSErr SPBOpenDevice  (ConstStr255Param deviceName, short permission,
                             long *inRefNum);
pascal OSErr SPBCloseDevice
                            (long inRefNum);
```

Recording Sounds Directly From Sound Input Devices

```
pascal OSErr SPBRecord      (SPBPtr inParamPtr, Boolean asynchFlag);
pascal OSErr SPBRecordToFile
                            (short fRefNum, SPBPtr inParamPtr,
                             Boolean asynchFlag);
pascal OSErr SPBPauseRecording
                            (long inRefNum);
pascal OSErr SPBResumeRecording
                            (long inRefNum);
pascal OSErr SPBStopRecording
                            (long inRefNum);
pascal OSErr SPBGetRecordingStatus
                            (long inRefNum, short *recordingStatus,
                             short *meterLevel,
                             unsigned long *totalSamplesToRecord,
                             unsigned long *numberOfSamplesRecorded,
                             unsigned long *totalMsecsToRecord,
                             unsigned long *numberOfMsecsRecorded);
```

Manipulating Device Settings

```
pascal OSErr SPBGetDeviceInfo
                            (long inRefNum, OSType infoType,
                             char *infoData);
pascal OSErr SPBSetDeviceInfo
                            (long inRefNum, OSType infoType,
                             char *infoData);
```

Constructing Sound Resource and File Headers

```
pascal OSErr SetupSndHeader
                            (Handle sndHandle, short numChannels,
                             Fixed sampleRate, short sampleSize,
                             OSType compressionType, short baseFrequency,
                             unsigned long numBytes, short *headerLen);
pascal OSErr SetupAIFFHeader
                            (short fRefNum, short numChannels,
                             Fixed sampleRate, short sampleSize,
                             OSType compressionType,
                             unsigned long numBytes,
                             unsigned long numFrames);
```

Registering Sound Input Devices

```
pascal OSErr SPBSignInDevice
                            (short deviceRefNum,
                             ConstStr255Param deviceName);
pascal OSErr SPBGetIndexedDevice
                            (short count, Str255 deviceName,
                             Handle *deviceIconHandle);
pascal OSErr SPBSignOutDevice
                            (short deviceRefNum);
```

Converting Between Milliseconds and Bytes

```
pascal OSErr SPBMilliSecondsToBytes
                            (long inRefNum, long *milliseconds);
pascal OSErr SPBBytesToMilliSeconds
                            (long inRefNum, long *byteCount);
```

Obtaining Information

```
pascal NumVersion SPBVersion
                            (void);
```

Application-Defined Routines

```
pascal void MySICompletionRoutine
                              (SPBPtr inParamPtr);
pascal void MySIInterruptRoutine
                              (void);
```

Assembly-Language Summary

Data Structures

Sound Input Parameter Block Data Structure

0	`inRefNum`	long	The input device reference number
4	`count`	long	The number of bytes to record
8	`milliseconds`	long	The number of milliseconds to record
12	`bufferLength`	long	The length of the buffer
16	`bufferPtr`	long	The address of the buffer
20	`completionRoutine`	long	A pointer to a completion routine
24	`interruptRoutine`	long	A pointer to an interrupt routine
28	`userLong`	long	For application's use
32	`error`	word	The error value returned after recording
36	`unused1`	long	Reserved

Trap Macros

Trap Macros Requiring Routine Selectors

_SoundDispatch

Selector	Routine
$00000014	SPBVersion
$01100014	SPBSignOutDevice
$021C0014	SPBCloseDevice
$02280014	SPBPauseRecording
$022C0014	SPBResumeRecording
$02300014	SPBStopRecording
$030C0014	SPBSignInDevice
$03200014	SPBRecord
$04240014	SPBRecordToFile
$04400014	SPBMillisecondsToBytes
$04440014	SPBBytesToMilliseconds
$05140014	SPBGetIndexedDevice
$05180014	SPBOpenDevice
$06380014	SPBGetDeviceInfo
$063C0014	SPBSetDeviceInfo
$07080014	SndRecordToFile
$08040014	SndRecord
$0B4C0014	SetupAIFFHeader
$0D480014	SetupSndHeader
$0E340014	SPBGetRecordingStatus

Result Codes

noErr	0	No error
abortErr	–27	Asynchronous recording was cancelled
permErr	–54	Attempt to open locked file for writing
userCanceledErr	–128	User canceled the operation
siNoSoundInHardware	–220	No sound input hardware available
siBadSoundInDevice	–221	Invalid sound input device
siNoBufferSpecified	–222	No buffer specified
siInvalidCompression	–223	Invalid compression type
siHardDriveTooSlow	–224	Hard drive too slow to record
siInvalidSampleRate	–225	Invalid sample rate
siInvalidSampleSize	–226	Invalid sample size
siDeviceBusyErr	–227	Sound input device is busy
siBadDeviceName	–228	Invalid device name

`siBadRefNum`	–229	Invalid reference number
`siInputDeviceErr`	–230	Input device hardware failure
`siUnknownInfoType`	–231	Unknown type of information
`siUnknownQuality`	–232	Unknown quality