Technote 1124

New Sound Input Driver Features

CONTENTS

This Technote describes the visible features and changes of the new PCI sound input drivers introduced with Mac OS 8.1.

This Note is directed at developers whose applications use sound input, and to a lesser extent sound output (playback). Developers who are creating sound input drivers should also scan through this Technote to see if these changes should be mirrored in their drivers.

Background of the Changes

Developers have asked for faster, more flexible sound input drivers with more useful features. The new PCI sound input drivers are the first step towards addressing these requests.

One request was for reduced latency between when the recording starts and when the application gets the first bit of recorded data.

Another request was for an interrupt buffer that was an integer power of 2 so that algorithms such as fast Fourier transforms could work more efficiently.

Users requested a simpler sound input model; existing drivers made it too hard to configure the Mac for simple audio tasks, such as listening to a CD or adjusting the computer's volume. Apple's support representatives frequently answered calls from users who were unable to listen to an audio CD. This problem is addressed by the new sound input drivers as well.

The final change, the removal of the sound input driver's interface (the `siOptionsDialog`), was requested by engineering so that sound input drivers could more closely conform to standard driver restrictions (like not calling `ModalDialog`).

Note:
Do not rely on checking the Mac OS version to determine which Sound Manager features are available. Update your code as if these changes were universal. While the sound input drivers changed for Mac OS 8.1, these changes could be applied to older versions of the Mac OS through an update to the Sound Manager system extension.

The Changes

Latency Reduction

One of the most common requests was for reduced interrupt latency in sound input and output (playback). This has been addressed by reducing the size of the hardware interrupt buffer from 1056 samples to 512 samples when virtual memory is off. When VM is on, the buffer size is 4096 samples, which is a reduction from its previous value of 4224 samples.

The number 512 was chosen for two reasons. By reducing the size of the buffer by approximately 50%, interrupt latency was reduced by 50% as well. Now sound interrupts happen approximately every 11 milliseconds instead of every 22 milliseconds (for 44.1KHz sound). This allows recorded data to be given to the application with less latency and allows for sounds being played to start sooner.

The second reason the buffer size was reduced to 512 samples is that 512 is a integer power of 2, which is important to developers trying to do real time analysis of the sound being recorded. The algorithms that are used to do this type of analysis (i.e., fast Fourier transforms) frequently require buffers that are an integer power of two. By adjusting the sound driver's buffer size, the developer no longer has to do any complicated buffering of data in their interrupt callback routine. Use the `siHardwareFormat` selector to check the playback hardware buffer size and `siDeviceBufferInfo` to check the input hardware buffer size.

Options Dialog Removed

The `siOptionsDialog` selector has been removed from the PCI sound input driver. It was removed for two reasons:

1. Drivers should not have a user interface (in fact, native device drivers ('ndrv') are not permitted to call `ModalDialog`).

2. Dialogs were not consistent between different Macintosh computer models.

It was felt that developers could provide a better interface than the `siOptionsDialog` interface, either through their own creation or the use of QuickTime's SequenceGrabber dialog.

New Ways to Select an Input Source

Developers asked for a uniform way to select sound input sources. This has been accomplished with the `siOSTypeInputSource` and `siOSTypeInputAvailable` selectors. Developers can now choose an input source by a standard `OSType` rather than by its name (which often changes across systems and gets localized).

Simpler Sound Model

Another change which helps to simplify the sound model on the Macintosh is that separate volume controls for the headphones and internal speaker have been merged into one control. There is no longer support for setting the headphone volume independently of the internal speaker's volume (because users were often confused by which volume slider they needed to adjust to get the proper volume level), or for controlling the mute state of the internal speaker when headphones are plugged in.

How to Gracefully Cope With These Changes

Latency Reduction

Dealing with the driver's input buffer change should not present any problems. For the most part, applications won't notice that anything has changed. The few applications that do care about the interrupt buffer size should already be calling `SPBGetDeviceInfo` with the `siDeviceBufferInfo` selector to get the size of the interrupt buffer. Correct code will adjust to the new buffer size without any modifications.

Options Dialog Removed

The removal of the `siOptionsDialog` selector requires the application developer to create an interface to properly configure the sound input driver for recording. The developer can no longer call `siOptionsDialog` or rely on some other software to configure the sound input driver.

There seems to be a lot of confusion on the point of how to properly configure the sound input driver. Keep these simple rules in mind when designing your application:

1. Never assume that the sound input driver retains its settings after being closed.
2. Never rely on some other program to set the sound input source for you.
3. Never rely on some other program to set the playthrough state for you.
4. Never assume the state of the sound input driver.

Under these simple rules, it is clear that relying on the user to set the sound input source through Monitors & Sound or the Sound control panel is the wrong thing to do: it breaks the first three rules.

The correct way to configure the sound input driver is to present an interface of your own creation, or QuickTime's SequenceGrabber interface, to the user. The minimum required interface must allow the user to select the desired input source and playthrough state. Once the input source and playthrough state (sample rate, sample size, number of channels, etc.) is set, do not close the input driver, as there is no guarantee that when the driver is reopened that the driver will default to the settings the user instructed your program to make. If you must close the input driver, reset it to the chosen configuration when you reopen it.

New Ways to Select an Input Source

To make it easier for application writers to choose the correct input source the new PCI sound input drivers implement two new selectors that work with OSTypes for selecting the input source. These new selectors are:

- `siOSTypeInputSource = FOUR_CHAR_CODE('inpt')`
- `siOSTypeInputAvailable = FOUR_CHAR_CODE('inav')`

The `siOSTypeInputSource` selector allows you set the input source by passing an `OSType` rather than an input source number. This is very useful considering the fact that same input source, such as the external microphone, could have radically different source numbers depending on what Mac you are running on. The `siOSTypeInputSource` selector currently uses these constants to specify an input source (from Universal Headers 3.1):

```
enum {
    kNoSource                = FOUR_CHAR_CODE('none'),
    kCDSource                = FOUR_CHAR_CODE('cd  '),
    kExtMicSource            = FOUR_CHAR_CODE('emic'),
    kRCAInSource             = FOUR_CHAR_CODE('irca'),
    kTVFMTunerSource         = FOUR_CHAR_CODE('tvfm'),
    kDAVInSource             = FOUR_CHAR_CODE('idav'),
    kIntMicSource            = FOUR_CHAR_CODE('imic'),
    kMediaBaySource          = FOUR_CHAR_CODE('mbay'),
    kModemSource             = FOUR_CHAR_CODE('modm'),
    kZoomVideoSource         = FOUR_CHAR_CODE('zvpc')
};
```

The `siOSTypeInputSource` selector can be used with both `SPBSetDeviceInfo` and `SPBGetDeviceInfo`, while `siOSTypeInputAvailable` can only be used with the `SPBGetDeviceInfo` call as it returns the list of OSType selectors that can be used on the current Mac.

For more information about how to use the `siOSTypeInputSource` and `siOSTypeInputAvailable` selectors see the Sample Code section and Q&A SND12.

You can always attempt to use these selectors without doing any Sound Manager/Sound Input Manager version checking beforehand. If an error is returned when you attempt to use these selectors, the sound input driver doesn't support these new selectors and you will have to have a fallback plan (like putting up a dialog for the user to select an input source) for setting the sound input source.

Simpler Sound Model

The removal of independent volume controls for the headphones and internal speaker was done to give users an interface more consistent with other audio hardware, such as home stereos. In this audio model, when headphones are plugged in, the main unit's speakers are disabled and the volume control now controls the headphone volume. It's a simple concept, like the one button mouse: with only one volume control you always know which one to use. Any API calls which affect the volume of the internal speaker also set the volume of the headphones and vice versa. There is no way to adjust the volumes independently; that feature has been removed.

This also allows the sound model to remain consistent with Macintoshes that physically don't allow for independent volume control of the headphones and internal speakers or which physically disconnect the internal speaker when headphones are plugged in.

Sample Code

Here is some sample code showing how to access the `OSType` array that is returned from `siOSTypeInputAvailable`.

```
static OSErr GetSoundInputSourceNames (long siRefNum, Handle *sourceNames)
{
    OSErr                   err;

// siInputSourceNames will return a structure that looks like:
//  struct {
//      short   numNames;
//      PString names[numNames];    // <<== these are packed pascal strings
//  };

    if (sourceNames != nil) {
        err = SPBGetDeviceInfo (siRefNum, siInputSourceNames, sourceNames);
    }

#if DEBUG
    if (err == noErr)
    {
        char                    sourceName[255];
        long                    offset;
        short                   numNames;
        int                     i;

        printf ("\nThe sound input source names are:\n");

        numNames = (**(short***)sourceNames)[0];

        offset = 2;
        for (i = 0; i < numNames; i++)
        {
            SInt32 strLen = ((char*)(**sourceNames))[offset++];

            BlockMoveData (&((char*)(**sourceNames))[offset],
                               sourceName, strLen);
            sourceName[strLen] = 0;
            printf ("  %s\n", sourceName);
            offset += strLen;
        }
    }
#endif

    return err;
}
```

Here is a sample illustrating a basic technique to create your own sound input options dialog. First, get the list of sound input source names and turn that list into a menu (in this case a pop-up menu), insert that menu into a dialog and then display the dialog. Once the dialog has been dismissed, if the user did not cancel it, you use `GetControlValue` to know which menu item they selected and then pass this value with the `siInputSource` selector to have the sound input driver use that input source. No re-mapping of the value from the menu is necessary because the sound input driver returns the list of names in the same order as `siInputSource` selects input sources.

```
#define kSourceNamesMenu        3
#define kPlayThruCheckBox       4
OSErr    DoSoundInputConfig (long soundInputDevice) {
    OSErr                   err                 = noErr;
    Rect                    box;
    Handle                  sourceNames;
    MenuHandle              namesMenu;
    ControlHandle           control;
    DialogPtr               optionsDialog;
    long                    offset,
                            i;
    short                   type,
                            inputSource,
                            itemHit,
                            playThruState;
    Boolean                 done                = false;

    err = GetSoundInputSourceNames (soundInputDevice, &sourceNames);

    namesMenu = NewMenu (kNamesMenu, nil);

    if (namesMenu != nil) {
        offset = sizeof (short);    // skip over count field
        for (i = 0; i < (*(short**)sourceNames)[0]; i++) {
            AppendMenu (namesMenu, &((unsigned char*)(*sourceNames))[offset]);
            offset += (*(char**)sourceNames)[offset] + 1;
        }

        InsertMenu (namesMenu, hierMenu);

        optionsDialog = GetNewDialog (kOptionsDialog, nil, (WindowPtr)-1L);

        GetDialogItem (optionsDialog, kSourceNamesMenu, &type, &
                      (Handle)control, &box);
        err = SPBGetDeviceInfo (soundInputDevice, siInputSource, &inputSource);
        SetControlValue (control, inputSource);

        GetDialogItem (optionsDialog, kPlayThruCheckBox, &type, &
                      (Handle)control, &box);
        err = SPBGetDeviceInfo (soundInputDevice, siPlayThruOnOff, &playThruState);
        if (playThruState > 1)
            playThruState = 1;
        SetControlValue (control, playThruState);

        SetDialogDefaultItem (optionsDialog, ok);
        ShowWindow (optionsDialog);

        while (done == false) {
            ModalDialog (nil, &itemHit);

            switch (itemHit) {
                case ok:
                    err = noErr;
                    done = true;
                    break;
                case cancel:
                    err = userCanceledErr;
                    done = true;
                    break;
                case kPlayThruCheckBox:
                    type = chkCtrl;
                    GetDialogItem (optionsDialog, kPlayThruCheckBox, &
                                   type, &(Handle)control, &box);
                    SetControlValue (control, !GetControlValue (control));
                    break;
            }
        }

        if (err == noErr) {
            GetDialogItem (optionsDialog, kSourceNamesMenu, &
                           type, &(Handle)control, &box);
            inputSource = GetControlValue (control);
            err = SPBSetDeviceInfo (soundInputDevice, siInputSource, &inputSource);

            if (err == noErr) {
                GetDialogItem (optionsDialog, kPlayThruCheckBox, &
                               type, &(Handle)control, &box);
                playThruState = GetControlValue (control);
                if (playThruState == 1)
                    playThruState = 7;
                err = SPBSetDeviceInfo (soundInputDevice, siPlayThruOnOff,
                                        &playThruState);
            }
        }

        DisposeDialog (optionsDialog);
        DisposeMenu (namesMenu);
    }

    return (err);
}
```

Additional Notes & Comments

As always, QuickTime exists to make dealing with audio (either playing or recording) easier. If your application only requires simple recording, you may wish to use QuickTime to do that recording. QuickTime supplies a standard user interface that can be accessed via one simple call, `SGSettingsDialog`, and QuickTime allows applications a simple way to record sound without having to know anything about the hardware the application is running on.

Summary

The changes to the PCI sound input drivers do not have to mean pain and suffering for you and your users if your application allows the user to configure the sound input driver from your application. However, if your application relies on other software to configure the sound input driver for you, now is the time to update your application to correctly handle all sound configuration itself.

Further References

- Technote 1108: Unknown Sound Features
- Technote 1048: Some Sound Advice: Getting the Most Out of the Sound Manager
- Inside Macintosh: Sound
- Inside Macintosh:QuickTime

Downloadables

Acrobat version of this Note (40K)

To contact us, please use the Contact Us page.
Updated: 7-Feb-2000

Technotes
Previous Technote | Contents | Next Technote