Technical Note TN2016

iTunes Visual Plug-ins

CONTENTS

This Note is directed at application developers who want to create visual plug-ins for iTunes 1.1 and later versions. (This feature is not supported in iTunes 1.0.)

If you received this documentation as part of the "iTunes Visual plug-ins SDK" package, you should check on Apple's web site for an updated version.

All of the code contained in this documentation can be found in the iTunes Visual plug-ins Sample Code provided in the SDK which is available on the sdk page at Apple's web site.

We'd like to hear about your finished Visual plug-in. Send information about it, and the Visual plug-in itself if you like, to itunesvisuals@mac.com.

[Jun 26 2001]

Overview

When the user clicks on the Visual Effects button in iTunes, custom visual special effects appear.

Figure 1. Visual Effects button

For ease of implementation across different architectures, iTunes plug-ins are shared libraries that export a single entry point. In addition, plug-ins do not link against the iTunes application. Instead, a callback function pointer is provided to your plug-in during initialization, and the iTunes API calls are implemented via that function.

iTunes sends messages to your plug-in to tell it when to initialize and clean up, when the user turns visual effects on and off, when the window is resized, when the user starts or stops playing music, etc. When visual effects are on, iTunes will forward keyboard, mouse, and update events for the visual effects window to your plug-in. Messages are described in more detail below.

While music is playing, iTunes sends render messages that include waveform data (sound samples) corresponding to the music that is currently playing, and a spectrum analysis of the samples. During registration, your plug-in indicates the number of channels of waveform and spectrum data it wants, and how often it wants to receive render messages.

During registration, your plug-in indicates if it wants to receive idle messages. Idle messages are sent periodically whether music is playing or not. When the music stops, you can draw in response to idle messages to fade out your visual effects smoothly.

Your plug-in can have preferences that get stored in the iTunes preferences file. The iTunes API includes functions to access your plug-in's preferences. iTunes sends the configure message when the user clicks on the options button to tell you to show your settings dialog.

All messages are sent at system task level, the level at which most application code runs. The functions in the iTunes API must also be called from system task level. Calling them from any other level, e.g., from a preemptive task or deferred task level, is not supported. See Technote 1104 for more info about execution levels.

Back to top

Plug-in discovery and registration

iTunes recursively scans the plug-ins folder, which is located beside the iTunes application itself on Mac OS 9, and at ~/Library/iTunes/iTunes Plug-ins/ (and /Library/iTunes/iTunes Plug-ins/ if that directory exists) on Mac OS X, and only that folder, for plug-ins. On Mac OS 9, iTunes recognizes shared library files of type 'hvpl' as visual plug-ins. On Mac OS X, iTunes recognizes plug-ins packaged as bundles with `CFBundlePackageType` 'hvpl'. That means that you have to distribute the same plug-in in two different formats, one as a shared library and one as a bundle, if you want it to be recognized by iTunes on both Mac OS 9 and Mac OS X. We recommend using iTunes' creator "hook" as the file or bundle signature so it gets an icon and kind string consistent with the other iTunes plug-ins.

A visual plug-in sample code is provided both in Mac OS 9 and Mac OS X formats in the iTunes Visual Plug-in SDK.

A plug-in library exports a single entry point. The name of the exported function depends on how the plug-in is packaged. If the plug-in is a single file CFM shared library, the entry point must be the CFM main entry point and the function name does not matter. If a bundled plug-in is CFM based, the name of the entry point must be "iTunesPluginMain". If a bundled plug-in is Mach-O, the name of the entry point must be "iTunesPluginMainMachO".

For each plug-in found, iTunes sends a `kPluginInitMessage` message to the entry point mentioned above, which is referred to as the plug-in "main" entry point in this SDK. A single plug-in file or bundle can contain multiple plug-ins. To support this, the plug-in main entry point must register each of the contained plug-ins when it receives the `kPluginInitMessage`, which will be sent only once.

To register a visual plug-in, you call `PlayerRegisterVisualPlugin`, passing a pointer to your visual plug-in message handler function. All `kVisualPlugin...` messages are sent to this handler function, starting with `kVisualPluginInitMessage`, which is sent immediately upon registration.

It is not necessary to unregister visual plug-ins. When it's time to shut down, a `kVisualPluginCleanupMessage` will be sent to your visual plug-in message handler, and then a `kPluginCleanupMessage` will be sent your plug-in's main entry point.

For compatibility with future versions of iTunes, if your plug-in receives a message it does not recognize, it should return `unimpErr`.

Back to top

Messages in Detail

1. Plug-in main entry point messages

These messages are sent to your plug-in main entry point, which has the following signature:

```
OSStatus main(OSType message, PluginMessageInfo * messageInfo, void * refCon);
```

`message`

what message is being sent

`messageInfo`

a pointer to additional parameters, if any.

`refCon`

the value you returned in `PluginInitMessage.refCon` will be passed back in this parameter.

This table lists the plug-in main entry point messages and corresponding `PluginMessageInfo` variants. Messages that have no additional parameters are listed as "n/a"; you should ignore `messageInfo` for those messages.

message	**messageInfo->u**
kPluginInitMessage	PluginInitMessage
kPluginCleanupMessage	n/a
kPluginIdleMessage	n/a

kPluginInitMessage

This message is sent to your plug-in library at launch time. You should register your plug-ins by calling `PlayerRegisterVisualPlugin`, described below, when you get this message. Then fill in the `options` and `refCon` fields of the `PluginInitMessage` before returning.

`messageInfo->u` is a `PluginInitMessage`:

```
struct PluginInitMessage {
 UInt32       majorVersion; /* Input */
 UInt32       minorVersion; /* Input */

 void *       appCookie;    /* Input */
 ITAppProcPtr appProc;      /* Input */

 OptionBits   options;      /* Output */
 void *       refCon;       /* Output */
};
```

`majorVersion minorVersion`

the version of the iTunes API implemented by the iTunes application.

`appCookie appProc`

parameters to be passed to the callback APIs

`options`

the currently defined options are intended for use by device plug-ins. Visual plug-ins should set this field to zero

`refCon`

the value you return here will be passed back as the `refCon` parameter in future calls to your main entry point

`kPluginCleanupMessage`

This message is sent to your plug-in when iTunes is about to quit.

`kPluginIdleMessage`

This message is intended for use by device plug-ins. Visual plug-ins that want idle time register for `kVisualPluginIdleMessage`, described below.

2. Visual plug-in messages

A visual plug-in message handler has the following signature:

```
OSStatus VisualPluginHandler(OSType message,
  VisualPluginMessageInfo * messageInfo, void * refCon);
```

`message`

what message is being sent.

`messageInfo`

a pointer to additional parameters, if any.

`refCon`

the value you returned in `VisualPluginInitMessage.refCon` will be passed back in this parameter.

This table lists the visual plug-in messages and corresponding `VisualPluginMessageInfo` variants. Messages that have no additional parameters are listed as "n/a", you should ignore `messageInfo` for those messages.

message	**messageInfo->u**
kVisualPluginInitMessage	VisualPluginInitMessage
kVisualPluginCleanupMessage	n/a
kVisualPluginIdleMessage	n/a
kVisualPluginConfigureMessage	n/a
kVisualPluginEnableMessage	n/a
kVisualPluginDisableMessage	n/a

kVisualPluginShowWindowMessage	VisualPluginShowWindowMessage
kVisualPluginHideWindowMessage	n/a
kVisualPluginSetWindowMessage	VisualPluginSetWindowMessage
kVisualPluginRenderMessage	VisualPluginRenderMessage
kVisualPluginUpdateMessage	n/a
kVisualPluginPlayMessage	VisualPluginPlayMessage
kVisualPluginChangeTrackMessage	VisualPluginChangeTrackMessage
kVisualPluginStopMessage	n/a
kVisualPluginSetPositionMessage	VisualPluginSetPositionMessage
kVisualPluginPauseMessage	n/a
kVisualPluginUnpauseMessage	n/a
kVisualPluginEventMessage	VisualPluginEventMessage

kVisualPluginInitMessage

This message is sent right after you register your plug-in with `PlayerRegisterVisualPlugin`. For this message only, the `refCon` parameter will be the value you returned in `PlayerRegisterVisualPluginMessage.registerRefCon`. Fill in the `options` and `refCon` fields of the `VisualPluginInitMessage` before returning.

`messageInfo->u.visualPluginInitMessage` is a `VisualPluginInitMessage`:

```
struct VisualPluginInitMessage {
 UInt32       messageMajorVersion; /* Input */
 UInt32       messageMinorVersion; /* Input */
 NumVersion   appVersion;          /* Input */

 void *       appCookie;           /* Input */
 ITAppProcPtr appProc;             /* Input */

 OptionBits   options;             /* Output */
 void *       refCon;              /* Output */
};
```

`messageMajorVersion`
`messageMinorVersion`

the version of the iTunes API implemented by the iTunes application.

`appVersion`

the version of iTunes that is running.

`appCookie`
`appProc`

The plug-in should copy these two fields into its private data since it will need to pass them as parameters to all the iTunes APIs described below.

`options`

currently there are no options defined, so your plug-in must set this to zero.

`refCon`

the value returned in this field will be passed as the `refCon` parameter in subsequent calls to the visual plug-in handler. Your plug-in can use this for anything it chooses, typically to store a pointer to data which is allocated in the init message. The sample code uses this technique to recover the pointer to its private data in its visual plug-in handler.

kVisualPluginCleanupMessage

This message is sent when iTunes is about to quit. You should free any resources allocated by your visual plug-in at this time.

kVisualPluginIdleMessage

This message is sent periodically if the plug-in requests idle messages. Do this by setting the `kVisualWantsIdleMessages` option in the `PlayerRegisterVisualPluginMessage.options` field.

kVisualPluginConfigureMessage

This message is sent when the user clicks on the Options button at the top right of the iTunes window. Enable the Options button (and this message) by setting the `kVisualWantsConfigure` option in the `PlayerRegisterVisualPluginMessage.options` field. The sample code shows how to implement a settings dialog for configuration, incuding how to access your resource fork and when to call `PlayerHandleMacOSEvent` to handle events.

kVisualPluginEnableMessage

kVisualPluginDisableMessage

iTunes currently enables all loaded visual plug-ins. Your plug-in should simply return `noErr` for these messages.

kVisualPluginShowWindowMessage

Sent when visual effects are turned on. At this point, the plug-in should allocate any large buffers it needs.

`messageInfo->u.showWindowMessage` is a `VisualPluginShowWindowMessage`:

```
struct VisualPluginShowWindowMessage {
 CGrafPtr   port;     /* Input */
 Rect       drawRect; /* Input */
 OptionBits options;  /* Input */
};
```

`port`

the port to draw into. The plug-in should remember this since it is not sent with render or update messages.

`drawRect`

the rect to draw into. The plug-in should remember this since it is not sent with render or update messages.

`options`

the only option currently defined is `kWindowIsFullScreen`, it's set when you are in full screen mode.

kVisualPluginHideWindowMessage

This message is sent when visual effects are turned off. Your plug-in should free any large buffers allocated in the course of rendering here.

kVisualPluginSetWindowMessage

This message is sent when the user resizes the iTunes window or toggles full screen mode. It is conceptually the same as a `kVisualPluginHideWindowMessage` followed by a `kVisualPluginShowWindowMessage` with the new window info, but your plug-in may be able to handle this combined message more efficiently.

kVisualPluginRenderMessage

This message is sent periodically when music is playing, at a rate specified during registration.

`messageInfo->renderMessage` is a `VisualPluginShowWindowMessage`:

```
struct VisualPluginRenderMessage {
 RenderVisualData * renderData;  /* Input */
 UInt32             timeStampID; /* Input */
};

struct RenderVisualData {
 UInt8 numWaveformChannels;
 UInt8 waveformData[kVisualMaxDataChannels][kVisualNumWaveformEntries];

 UInt8 numSpectrumChannels;
 UInt8 spectrumData[kVisualMaxDataChannels][kVisualNumSpectrumEntries];
};
```

`renderData->numWaveformChannels`

The number of channels of waveform data included. This will be the number you requested during registration.

`renderData->waveformData`

The most significant 8 bits of the sound samples that are currently playing. The values range from 0 to 255, where 128 is the midpoint (AC zero value).

`renderData->numSpectrumChannels`

The number of channels of spectrum data included. This will be the number you requested during registration.

`renderData->spectrumData`

This is a 512-point Fast Fourier Transform of the waveform data.

kVisualPluginUpdateMessage

This message is sent in response to an update event. The visual plug-in should update into its remembered port. This will only be sent if the plug-in's window is showing, i.e. in between `kVisualPluginShowWindowMessage` and `kVisualPluginHideWindowMessage` messages.

kVisualPluginPlayMessage

This message is sent when iTunes starts playing a track. Your plug-in should copy any track info it wants to display.

`messageInfo->u.playMessage` is a `VisualPluginPlayMessage`:

```
struct VisualPluginPlayMessage {
 ITTrackInfo *      trackInfo;   /* Input */
 ITStreamInfo *     streamInfo;  /* Input */
 SInt32             volume;      /* Input */

 UInt32             bitRate;     /* Input */

 SoundComponentData soundFormat; /* Input */
};
```

kVisualPluginChangeTrackMessage

This message is sent when the information about a track changes, e.g., when the user edits track info, or when iTunes begins playing a different track. Your plug-in should copy any track info it wants to display.

`messageInfo->u.changeTrackMessage` is a `VisualPluginChangeTrackMessage`:

```
struct VisualPluginChangeTrackMessage {
 ITTrackInfo *      trackInfo;   /* Input */
 ITStreamInfo *     streamInfo;  /* Input */
};
```

kVisualPluginStopMessage

This message is sent when the music stops playing.

kVisualPluginSetPositionMessage

This message is sent when iTunes changes the elapsed time position within a track. A plug-in that shows the elapsed time would use this.

`messageInfo->u.setPositionMessage` is a `VisualPluginSetPositionMessage`:

```
struct VisualPluginSetPositionMessage {
 UInt32 positionTimeInMS; /* Input */
};
```

`kVisualPluginPauseMessage kVisualPluginUnpauseMessage`

iTunes does not currently use pause or unpause. A pause in iTunes is handled by stopping and remembering the position. Your plug-in should simply return `noErr` for these messages.

kVisualPluginEventMessage

This message is sent when the user generates an event that could be handled by your plug-in. If your plug-in handles the event, it should return `noErr`. Otherwise it should return `unimpErr`.

`messageInfo->u.eventMessage` is a `VisualPluginEventMessage`:

```
struct VisualPluginEventMessage {
 EventRecord * event; /* Input */
```

```
};
```

Back to top

Callback APIs

As mentioned in the overview, plug-ins do not link against iTunes. Instead, the first two parameters of all of the iTunes APIs are a cookie and a callback function pointer. These are provided to your plug-in in the init messages. The glue code in iTunesAPI.c takes care of marshaling parameters and calling iTunes for you, so you don't have to worry about the details.

```
OSStatus PlayerRegisterVisualPlugin (void *appCookie, ITAppProcPtr appProc,
   PlayerMessageInfo *messageInfo);
```

Register your visual plug-in with `PlayerRegisterVisualPlugin` when you receive `kPluginInitMessage`. Pass a `PlayerRegisterVisualPluginMessage` variant of `PlayerMessageInfo` as the `messageInfo` parameter. The fields are explained in the comments below:

```
    /* PlayerRegisterVisualPluginMessage.options */

enum {
        /* set to enable kVisualPluginIdleMessage */
    kVisualWantsIdleMessages = (1L << 3),

        /* set to enable kVisualPluginConfigureMessage */
    kVisualWantsConfigure    = (1L << 5)
};


struct PlayerRegisterVisualPluginMessage {
        /* Displayed in the Visual menu,
        also used to id prefs data*/
    Str63 name;
        /* See above enum */
    OptionBits options;

        /* Identifies the plug-in */
    OSType creator;

        /* Version number of the plug-in */
    NumVersion pluginVersion;

        /* Handler for the plug-in's messages */
    VisualPluginProcPtr handler;

        /* RefCon for the plug-in's handler */
    void *registerRefCon;

        /* How often to render, in milliseconds
        (0xFFFFFFFF = as often as possible) */
    UInt32 timeBetweenDataInMS;


    UInt32 numWaveformChannels; /* 0-2 waveforms requested */
    UInt32 numSpectrumChannels; /* 0-2 spectrums requested */

    SInt16 minWidth; /* Minimum resizeable width */
    SInt16 minHeight; /* Minimum resizeable height */

    SInt16 maxWidth; /* Maximum resizeable width */
    SInt16 maxHeight; /* Maximum resizeable height */

    UInt16 minFullScreenBitDepth;  /* 0 = Any */
    UInt16 maxFullScreenBitDepth;  /* 0 = Any */

        /* Reserved (should be zero) */
    UInt16 windowAlignmentInBytes;
};
```

```
OSStatus PlayerIdle (
    void *appCookie,
    ITAppProcPtr appProc);
```

Your plug-in should call `PlayerIdle` to give time to iTunes if it engages in a lengthy process.

```
void PlayerShowAbout(
    void *appCookie,
    ITAppProcPtr appProc);
```

Show the "About iTunes" window.

```
void PlayerOpenURL (
    void *appCookie,
    ITAppProcPtr appProc,
    SInt8 * urlString,
    UInt32 length);
```

Tell iTunes to open a URL. The `urlString` parameter is not a Pascal string or a C string; its length is specified by the `length` parameter.

```
OSStatus PlayerGetPluginData (
    void *appCookie,
    ITAppProcPtr appProc,
    void *dataPtr,
    UInt32 dataBufferSize,
    UInt32 *dataSize);
```

Read data identified by your plug-in's name from the iTunes preferences file. This can be used to store your plug-in's preferences as a single block of data. Your plug-in's name is specified in `PlayerRegisterVisualPluginMessage.name`.

```
OSStatus PlayerSetPluginData (
    void *appCookie,
    ITAppProcPtr appProc,
    void *dataPtr,
    UInt32 dataSize);
```

Save data identified by your plug-in's name from the iTunes preferences file. This can be used to store your plug-in's preferences as a single block of data. Your plug-in's name is specified in `PlayerRegisterVisualPluginMessage.name`.

```
OSStatus PlayerGetPluginNamedData (
    void *appCookie,
    ITAppProcPtr appProc,
    ConstStringPtr dataName,
    void *dataPtr,
    UInt32 dataBufferSize,
    UInt32 *dataSize);
```

Read data identified by `dataName` and your plug-in's name from the iTunes preferences file. This can be used to store your plug-in's preferences, with each preference field stored separately. Your plug-in's name is specified in `PlayerRegisterVisualPluginMessage.name`.

```
OSStatus PlayerSetPluginNamedData (
    void *appCookie,
    ITAppProcPtr appProc,
    ConstStringPtr dataName,
    void *dataPtr,
    UInt32 dataSize);
```

Save data identified by `dataName` and your plug-in's name from the iTunes preferences file. This can be used to store

your plug-in's preferences, with each preference field stored separately. Your plug-in's name is specified in `PlayerRegisterVisualPluginMessage.name`.

```
OSStatus PlayerHandleMacOSEvent (
    void *appCookie,
    ITAppProcPtr appProc,
    const EventRecord *theEvent,
    Boolean *eventHandled);
```

Ask iTunes to handle an event. The sample code uses this in its `SettingsDialogFilterProc`.

```
OSStatus PlayerGetPluginFileSpec(
    void *appCookie,
    ITAppProcPtr appProc,
    FSSpec *pluginFileSpec);
```

Return your plug-in's `FSSpec`. For bundled plug-ins, this will be the `FSSpec` of the bundle.

```
OSStatus PlayerSetFullScreen (
    void *appCookie,
    ITAppProcPtr appProc,
    Boolean fullScreen);
```

Tell iTunes to enter or exit full screen mode. If your plug-in wants to behave like a screen saver it could use this.

```
OSStatus PlayerSetFullScreenOptions (
    void *appCookie,
    ITAppProcPtr appProc,
    SInt16 minBitDepth,
    SInt16 maxBitDepth,
    SInt16 preferredBitDepth,
    SInt16 desiredWidth,
    SInt16 desiredHeight);
```

Specify bit depth and resolution for subsequent uses of full screen mode. If your plug-in has a user interface for setting these options, call this to update the values from what you specified in `PlayerRegisterVisualPluginMessage`.

Back to top

References

iTunes Visual plug-ins SDK package

iTunes Visual plug-ins Sample Code

Back to top

Downloadables

Acrobat version of this Note (136K) Download

Back to top