iMovie Plug-ins

(Legacy)

Apple Applications > iMovie

2007-09-04

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

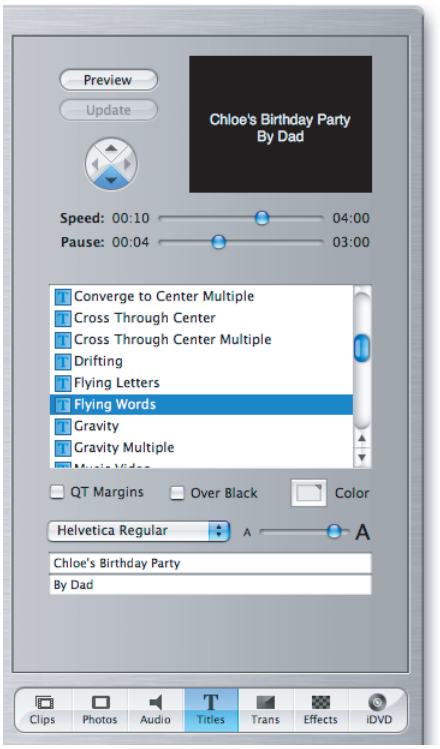
Contents

Tables and Listings

iMovie Plug-ins

Important: The information in this document is obsolete and should not be used for new development.

In addition to the basic movie editing capabilities such as cutting, adding, and rearranging frames, iMovie provides access to a number of enhancements such as titles, effects, and transitions. For example, in the Titles pane, the user can enter title text and pick from various title treatments, as shown below.



These enhancements are provided by plug-ins that use the plug-in architecture built into iMovie. For example, as shown in the figure above, the user might enter a movie title and select Flying Words. Clicking Preview would then cause iMovie to start the Flying Titles plug-in, which would then render the title frames in the iMovie monitor.

These enhancements are provided by plug-ins that use the plug-in architecture built into iMovie. For example, as shown in figure 1, the user might enter a movie title and select Flying Words. Clicking Preview would then cause iMovie to start the Flying Titles plug-in, which would then render the title frames in the iMovie monitor.

iMovie provides support for plug-ins as well as an application programming interface (API) that allows you to write your own plug-ins. The plug-in architecture allows you to take advantage of the built-in user interface elements such as directional buttons common to many plug-ins, and it allows you to add UI elements required by your plug-in.

At startup, iMovie finds and loads all valid plug-ins. After successfully loading a plug-in, iMovie communicates with it only when necessary. This communication starts when the user selects a service from the menu in the Titles, Transitions, or Effects pane .

From the perspective of the plug-in, this communication typically takes place during a session that includes three phases: initialization, processing, and termination. There can be multiple instances of a single plug-in (for example, rendering instances, in-line proxies in the iMovie that can be "played through", and preview instances). All three phases are performed for each plug-in instance.

Note that you can attach data to the instance pointer using the API, but be very careful about GLOBAL data because of re-entrancy issues (several instances may start before any of them finish).

The following sections provide details of how you write a plug-in to communicate properly with iMovie.

Plug-in Overview

A plug-in must be a bundle or shared library with a single entry point, which should be compiled as main. The main entry prototype is `HYPluginMainProcPtr`:

Listing 1-1 Main entry prototype

```
typedef HYPluginResult (*HYPluginMainProcPtr) (short which, HYPluginParams
*params);
```

iMovie passes the plug-in a selector in the selector field of the `HYPluginParams` structure. The plug-in must then perform the appropriate action for that selector. Three selectors of particular note are `kPlugInitialize`, `kPlugDoFrame`, and `kPlugTerminate`, which correspond respectively to the initialization, processing, and termination phases introduced above.

The `which` parameter identifies the item the user has chosen by name from the menu. Menu name functionality is described in detail in the section "Loading plug-ins" (page 8).

iMovie creates an opaque pointer, `HYPluginInstance`, that it uses to track the plug-in. If, for example, the user has created several simultaneously rendering effects based on a single plug-in, iMovie uses the `HYPluginInstance` to differentiate between the plug-in instances rendering those effects. The `HYPluginInstance` may be passed to the plug-in. The plug-in is expected to preserve it unchanged and possibly to return it in some of the API calls.

Loading plug-ins

When iMovie starts, it looks in the `iMovie Plugins` folder for plug-in bundles or files, as identified by their creator, `'Hway'` and type:

- `'titl'`, for title plug-ins
- `'trans'`, for transition plug-ins
- `'filt'`, for effects plug-ins

Constants for these creator and type values are as follows:

```
kPluginFileCreator ' Hway'
kPluginTitleFileType 'titl'
kPluginTransitionFileType 'tran'
kPluginFilterFileType 'fltr'
```

When your plug-in is identified by its type and creator, it must check its compatibility with the current version of iMovie. If it is not compatible, it must return an error code, upon receipt of which, iMovie will discontinue loading the plug-in. For further version information, see the section "API Release Versions" (page 14).

If the plug-in reports itself to be compatible, it must pass iMovie a string with one or more menu names to be presented in the menus to identify the services your plug-in can provide for the user. The name string is in `KResMenuNames`, a STR# resource with ID 16000.

Note that the ability to use multiple names allows you to provide multiple services within a single plug-in. If your plug-in takes advantage of this facility, it must be able to recognize any of its names and perform the service associated with that name.

As part of the calling sequence, the index of the selected menu name in the STR# resource is passed to the plug-in in the which parameter. The plug-in uses this index to identify and perform the appropriate service. For example, the services Flying Letters and Flying Words, shown in the menu in figure 1, are provided by the Flying Titles plug-in. These two services, then, correspond to indexes of 0, and 1, and the menu names for those services are retrieved from the STR# resource using those resource indices.

A STR# resource of ID 18000, identified by the constant `kResGroupNames`, must be created to provide for group names, to allow the future possibility of grouping related menu names into sub-menus. Although group names are not currently used, it is recommended that your plug-in provide this string. For example, you might want to provide a short version of your company name.

A STR# resource of ID 15000, identified by the constant `kResProgrammerNames`, must be created to allow for a file to be associated with the specific plug-in servicethat created it. See "Processing Phase" (page 11)for more detail.

Initialization Phase

Much of the communication between iMovie and your plug-in during the initialization phase is accomplished by passing settings information in a predefined *`HYPluginGetInfo`* structure. See "The HYPluginGetInfo Data Structure" (page 12), for more detail on this structure.

When the user selects a service from the menu in the Titles, Transitions, or Effects pane, iMovie calls the plug-in with the kPlugGetInfo selector. It is assumed that the plug-in has set the information in the HYPluginGetInfo data structure identified by HYPluginParams.getInfo. If, for some reason, the plug-in cannot run, it can return an error code and/or fill in 0 for the framesToBeRendered field of the HYPluginGetInfo structure.

A plug-in informs iMovie as to what type it is by setting HYPluginGetInfo.pluginType to one of the type constants shown in Table 1-1.

Table 1-1 Valid plug-in type constants

Constant name	Notes
`kPluginUnknownType`	Value: 0. Default value—no valid plug-in available.
`kPluginTitleType`	Value: 101. A title plug-in
`kPluginFilterType`	An effect plug-in
`kPluginTransitionBetweenType`	A transition performed between two clips
`kPluginTransitionBeforeType`	A transition performed before the current clip
`kPluginTransitionAfterType`	A transition performed after the current clip

A transition is considered to be one of three types: before, between, or after. A between-type transition, the most common type, is displayed between two clips. An example of a before-type transition is a fade-in; it is displayed before the clip it fades in (and doesn't require another clip in front of it). A fade-out is an example of an after-type transition.

iMovie then calls the plug-in with the `kPlugInitialize` selector. At this time the plug-in should set up any required data structures. The plug-in can attach arbitrary data to the HYPluginParams structure using the pointer in that structure's clientData field. That pointer is then returned with each subsequent call. The plug-in is responsible for destroying such data during the termination phase of the session.

Enabling User Interface Elements

Part of the initialization phase is to set the default state of the user interface (UI) elements in the plug-in pane. The plug-in sets plug-in choice flags (*`HYPluginGetInfo.pluginFlags`*) to specify user interface elements to be enabled to accept user data, settings, and selections. Plug-in choice flags applicable to the Title, Transition, and Effects panes are shown in Table 1-2. (Flags for UI elements specific to a plug-in type are described below in a section for that plug-in type.)

Table 1-2 Plug-in choice flag constants

Constant name	Notes
`kPluginChoiceVertical`	Value: `1L`. Enable up and down buttons
`kPluginChoiceHorizontal`	Value: `2L`. Enable left and right buttons
`kPluginChoiceSpeed`	Value: `4L`. Enable speed slider
`kPluginChoiceAll`	Value: `7L`. Enable up, down, left, right buttons and speed slider
`PLUGIN_COMBO_VERTICAL`	Value: `5L`. Enable up, down buttons and speed slider
`PLUGIN_COMBO_HORIZONTAL`	Value: `6L`. Enable left, right buttons and speed slider
`PLUGIN_COMBO_NOMOTION`	
`kPluginChoiceXYLocation`	Value: `1024L`. Enable X,Y coordinate selection.

Directional Arrows

The arrows in the directional button cluster on the left side in the Title, Transition, and Effects panes are enabled using the vertical and horizontal plug-in choice flags listed in Table 1-2 (page 10). They can be ORed together to achieve the desired combinations.

X,Y Location

If the plug-in enables the XYLocation flag, the iMovie small preview rectangle provides a cross-hair that the user can drag to specify an XY location. This feature was introduced in iMovie 2.1.1 and is ignored by prior versions

Processing Phase

The processing phase is when the plug-in does most of its work. iMovie controls the processing phase through a loop that calls the plug-in as many times as necessary, as specified by the value of *HYPluginGetInfo.framesToBeRendered*. These calls are made with the `kPlugDoFrame` selector, and with each call *HYPluginParams.renderFrame* is updated. The value of the first frame in the movie, *HYPluginParams.startFrame*, is expressed in global frame numbers. The value of *renderFrame* is a zero-based index relative to startFrame. This approach allows the plug-in to request frames relative to the current frame using the callbacks (for example, the previous frame would be *renderFrame*-1. The value of *HYPluginParams.frameCount* gives the plug-in the total number of frames to render. Thus, *renderFrame* can range from `0` to *frameCount*.

The plug-in can call back into iMovie using a variety of procedures that are provided through the *HYPluginParams.procs* pointer. These include procedures for drawing, working with frames, clips, and movies, and so forth. Memory allocation should be done through these procedure pointers, so that iMovie can keep track of plug-in memory use. See the HYPluginProcs.h header file for the details and prototypes.

Drawing

With each `kPlugDoFrame` call, iMovie passes a pointer (in *HYPluginParams.frame*) to a gWorld that contains the current (uncompressed) frame of video. The plug-in performs its service and draws into the gWorld before returning. For example, an effect might use the `GetFrame` callback to get the pixels for its starting frame, perform its calculations, and then draw into the gWorld before returning. In the less complicated example of a title, for which only the current frame is of interest, the plug-in might simply compute the X,Y location based on the frame number, and draw some text into the gWorld.

It is important to note that the plug-in may be asked to draw in an arbitrary GWorld with a rectangle. Consequently, the scale of text is not fixed, but should be computed relative to the size of the rectangle. For example, the plug-in may be called to preview its results in the large video window, in a small preview window, or even in a thumbnail during rendering. The plug-in must also not assume that the rectangle passed starts at 0,0

Any frame may be drawn at any time; the plug-in must be able to draw frame N without having drawn frame N-1. This is so that arrow keys may be used to go forward and back in Preview mode, for example.

The plug-in can call `GetFrame` with arbitrary frame numbers relative to startFrame, but the requested frames may or may not exist.

iMovie creates an opaque pointer, *HYFrameCookie*, to keep track of a frame as it goes to and from the plug-in. The plug-in is expected to preserve this pointer unchanged and possibly to return it in some of the API calls.

Working With Clips

iMovie passes back the *HYPluginClipInfo* structure when the plug-in makes the `GetClipInfo` callback. This structure allows the plug-in to determine settings for a clip such as the clip name, its audio volume, and whether it has a fade-in or fade-out. The plug-in can change some of such settings and then pass them back in with `SetClipInfo`.

The *HY2PluginClipInfo* structure was introduced in iMovie version 2.0 to extend *HYPluginClipInfo*. This structure contains settings for reverse play, frame speed, and length (in frames) of a fade-in or fade-out.

Note: There is currently no reliable way for the plug-in to find another whole clip.

Working With Thumbnails

The plug-in should set *HYPluginGetInfo.bestThumbOffset* to specify which frame is to be used for a thumbnail. Consider, for example, a title that scrolls on-screen; the first frame would contain no visible text. One simple approach might be to set *bestThumbOffset* to the middle frame (derived from *HYPluginGetInfo.frameCount*).

Working With Saved Files

The plug-in must specify a programmer name (as described above in the section "Initialization Phase") to allow iMovie to associate the plug-in with a rendered file (title, transition, filter) created by that plug-in. These names are stored in the iMovie project file and are also loaded into the record stored in the menu so the application can select the proper plug-in if the user selects an existing rendered file.

The HYPluginGetInfo Data Structure

The plug-in provides setup information to iMovie primarily through the HYPluginGetInfo data structure, which includes the fields shown in Table 1-3 (page 12).

Table 1-3 Fields in HYPluginGetInfo

Field Name	Notes
pluginType	Either a transition or a title
titleType	Either single-pair, multi-pair, or block
pluginFlags	Settings for speed, direction, font, color, etc.
framesToBeRendered	Plug-in returns to iMovie based on settings in UI

Field Name	Notes
takeFromClipBefore	Frames to be taken from previous clip for rendering
takeFromClipAfter	Frames to be taken from next clip for rendering
minSpeedFrames	Left endpoint of Speed slider
maxSpeedFrames	Right endpoint of Speed slider
mustTakeFromSingleClips	Allow/disallow consuming frames of multiple clips
bestThumbOffset	Best frame for making thumbnail
defaultColor	For when color choice is not supported
defaultDirection	Default direction (up, down, left, right)
justify	Justification for current direction settings
defaultFontNum	Default font
defaultFontSize	Not currently used

The size of this structure is fixed. The functionality of the structure has been extended through the addition of the `HY2PluginGetInfo` for version 2 of the API (as shown in Table 1-4 (page 13)), `HY3PluginGetInfo` for version 3 of the API, as shown in Table 1-5 (page 14), and the `HY4PluginGetInfo` for version 4 of the API, as shown in Table 1-6 (page 14). This approach allows existing plug-ins to continue to work without limiting the access of newer plug-ins to new features.

Table 1-4 Fields in HY2PluginGetInfo (Corresponds to version 2 of the iMovie API.)

Field name	Notes
group	Group name (128 characters maximum)
filterSliders[]	Structures for effect-related sliders (currently, a maximum of 3)
defaultFadeInFrames	Default number of frames over which to ease in an effect
defaultFadeOutFrames	Default number of frames over which to ease out an effect
minHangFrames	Left endpoint of Pause slider
maxHangFrames	Right endpoint of Pause slider
defaultFontSizeScale	From 0.0 to 1.0. (See note, below.)
selectedFrames	Number of frames selected for an effect
speedFrames	Plug-in returns to iMovie based on user's choices
hangFrames	Plug-in returns to iMovie based on user's choices
baseMemoryNeeded	Amount ofRAM allocated during initialization phase

Note: The value of `defaultFontSizeScale` represents the value of the `TitleFontSize` slider, not the actual scale factor. The plug-in must interpret this value in terms of actual font size.

Table 1-5 Fields in HY3PluginGetInfo (Corresponds to version 3 of the iMovie API.)

Field name	Notes
`defaultX`	Starting value on X axis
`defaultY`	Starting value on Y axis
`userX`	User-selectedvalue on X axis
`userY`	User-selected value on Y axis
`fileInfo`	FSSpec of user-selected file
`buttonTitle`	31-character title of custom button

Note: The X and Y values are coordinates in a 640 by 480 DV frame.

Table 1-6 Fields in HY4PluginGetInfo (Corresponds to version 4 of the iMovie API.)

Field name	Notes
`titleSliders`	Array of title slider information

Video Standards: PAL versus NTSC

Constants are defined in `HYRefDigitalVideoStandard` to differentiate between PAL and NTSC video streams: `kRefNTSCStandard`, and `kRefPALStandard` (as well as `kRefUnknownStandard`). The plug-in can use either one, but cannot mix them within a project.

Video Compression Quality

The DV standard calls for only a single quality level in compressed form, but DV can be decompressed into lower quality (faster) or higher quality (better) video. A plug-in might, for example, drive a small preview at lower quality to improve performance. `HYRefDVQuality` provides three constants used by some API calls to ask for different quality levels: `kDVLowQuality`, `kDVHighQuality`, `kDVHighQualitySingleField`.

API Release Versions

Extensions of the iMovie API have been designed so that all released versions of the iMovie API are backward and forward compatible. A plug-in should be able to claim to be built against version 1 of the interface, and adjust at runtime for changes in the API.

To enable this degree of flexibility, iMovie does not enforce version compatibility; the plug-in must determine whether it can run in a given version of iMovie. To this end, iMovie defines several version number constants (shown in Table 1-7), and it passes its version down to the plug-in.

Table 1-7 iMovie application and API version designations

iMovie	API	Constant	Value
1.0	1	`kPluginInterfaceVersion1`	`1`
2.0	2	`kPluginInterfaceVersion2` `kPluginInterfaceLastVersion`	`2` `2`
2.1.1	3	`kPluginInterfaceVersion` `kPluginInterfaceVersionString`	`3` `3.0`

3.0.1	4	`kPluginInterfaceVersion4`
		`kPluginInterfaceLastVersion`
		`kPluginInterfaceVersion`
		`kPluginInterfaceVersionString`
			`4`
			`3`
			`4`
			`4.0`

In case of a version mismatch, the application can return one of two error codes: `kPluginAppIsTooOld` or `kPluginAppIsTooNew`. In either case, the plug-in won't load.

Termination phase

The termination phase begins when iMovie passes in the `kPlugTerminate` selector for the plug-in instance. At this point it is assumed that the plug-in has completed all of its processing for this instance. The plug-in should perform all necessary clean-up and return the appropriate result code.

Result Codes

Table 1-8 lists the result codes (defined in `HYPluginResult`) that the plug-in can return, or that might be returned to the plug-in by many of the API procedures.

Table 1-8 Result codes in iMovie

Result code	Notes	Returned by
`kPluginOK`	Value: 0. Operation completed successfully.	iMovie, plug-in
`kPluginFinished`	Operation terminated unsuccessfully	iMovie, plug-in
`kPluginBadParams`	One or more parameters received were invalid	iMovie, plug-in
`kPluginOutOfMemory`	Plug-in couldn't get necessary RAM	iMovie

Result code	Notes	Returned by
kPluginOutOfDiskSpace	Target disk doesn't have room to write a file	iMovie
kPluginFailed	Operation terminated unsuccessfully	plug-in
kPluginCantGetNeededFrame	Plug-in couldn't obtain a frame required for its service	plug-in
kPluginUnimplemented	Selected service does not exist	iMovie
kPluginTryAgainLater	Selected service is not currently available	iMovie, plug-in
kPluginAppIsTooOld	Plug-in incompatible with older iMovie	plug-in
kPluginAppIsTooNew	iMovie incompatible with older plug-in	plug-in

Title Plug-ins

A title plug-in identifies itself as such to iMovie by setting *HYPluginGetInfo.pluginType* to `kPluginTitleType`. It then further identifies itself by setting *HYPluginGetInfo.titleType* to cause the UI in iMovie to adjust to provide field editors for one text pair (a title string and subtitle string), multiple text pairs, or a block of text. The following values are valid: `kTitleUnknown` (value 1), `kTitleSingleTextPair`, `kTitleMultiTextPair`, `kTitleTextBlock`.

When a plug-in calls `GetTitleInfo()` to ask iMovie for information about a title, iMovie returns an *HYPluginTitlePair* structure that represents the text of a title pair (a title and subtitle string):

Listing 1-2 HYPluginTitlePair structure

```
typedef struct _HYPluginTitlePair {
    char *title;
    char *subTitle;
} HYPluginTitlePair;
```

This structure may be NULL if both the title and subtitle are empty.

Enabling User Interface Elements

A title plug-in sets plug-in choice flags (*HYPluginGetInfo.pluginFlags*) to specify user interface elements to be enabled to accept user data, settings, and selections. Plug-in choice flags applicable to the Title pane are shown in Table 1-9.

Table 1-9 Ttile pane plug-in choice flag constants

Constant name	Notes
kTitleChoiceSize	Value `8L`. Enable font size slider.
kTitleChoiceFont	Value `16L`. Enable font popup menu.

Constant name	Notes
kTitleChoiceHangTime	Value 32L. Enable pause slider
TITLE_COMBO_STANDARD	Enable up, down, left, right buttons; font popup menu; and speed slider
TITLE_COMBO_VERTICAL	Enable up and down buttons, font popup menu, and speed slider
TITLE_COMBO_HORIZONTAL	Enable left and right buttons, font popup menu, and speed slider
kTitleChoiceNoScaling	Value 64L. Changes automatic scaling of frames from 720 x 480 pixels to 640 x 480 pixels. This should be used only for very specific situations.
kTitleChoiceAll	Value 127L. Enable all title-related UI elements

Obtaining User's Direction Choice

The *HYPluginDirection* enumeration defines five values for the user-selected direction:

- kPluginDirectionNone
- kPluginDirectionUp
- kPluginDirectionDown
- kPluginDirectionLeft
- kPluginDirectionRight

For example, if the user clicks the right arrow, iMovie provides kPluginDirectionRight value in *HYPluginTitleInfo.direction*. The plug-in can also supply *HYPluginGetInfo.defaultDirection* to iMovie so it can set up the arrow to the desired default when the plug-in is initially chosen.

Justifying Text

To align the text in the text edit box, the plug-in sets *HYPluginGetInfo.justify* to one of the following values:

- kTitleJustCenter
- kTitleJustLeft
- kTitleJustRight

The default value is kTitleJustLeft.

This setting is currently used only in the block style of text plug-in, so, for example, if the plug-in right-justifies the text when the kPluginDirectionRight direction is chosen, setting *HYPluginGetInfo.justify* to kTItleJustRight causes the user's editing to be in a right-justified text block. This approach is used, for example, in the Music Video plug-in.

The HYPluginTitleInfo Structure

The *HYPluginTitleInfo* structure is passed back when you call the `GetTitleInfo` callback. It provides a wealth of information about the text entered by the user, the options chosen in the Titles panel, and so forth. Note that some fields are valid only for certain types of plug-ins. For example, if *HYPluginGetInfo.pluginType* is `kPluginTitleType` and *HYPluginGetInfo.titleType* is `kTitleSingleTextPair`, then the *blockLen* and *textBlock* fields are not defined.

Listing 1-3 HYPluginTitleInfo structure

```
// HYPluginTitleInfo
typedef struct _HYPluginTitleInfo {
    RGBColor foreColor; // text color
    RGBColor shadowColor;   // shadow color
    Boolean overBlack;  // if over black
    Boolean tvSafe; // if TV Safe margins (and colors?) wanted
    short fontID;   // Mac font ID
    float speed;    // slow to fast, in range 0.0 - 1.0
    long duration;  // in frames
    HYPluginDirection direction;    // direction of motion for title

    // -- actual text data
    short pairCount;    // used for "Pair" style plug-ins
    HYPluginTitlePair *pairs;   // used for "Pair" style  plug-ins

    long blockLen;  // used for "Block" style plug-ins
    char *textBlock;    // used for "Block" style plug-ins

    HY2PluginTitleInfo ver2;
      // no changes in version 3
      HY4PluginTitleInfo ver4;

} HYPluginTitleInfo;
```

The *HY2PluginTitleInfo* structure was added with iMovie 2.0. It is used only to extend the *HYPluginTitleInfo* structure.

Listing 1-4 HY2PluginTitleInfo structure

```
typedef struct _HY2PluginTitleInfo {
    float fontSizeScale;    // 0.0->1.0 This value doesn't represent
        // the actual scale factor, just the
        // value of the TitleFontSize slider. It
        // is up to each plug-in to determine
        // this value in terms of actual font size
    float hangTime; // 0.0->1.0
    } HY2PluginTitleInfo;
```

Note that a title plug-in can apply its results across multiple clips. In such a case, iMovie applies all the resulting title frames to the stored movie in a single clip, effectively replacing those clips, and thereby reducing the number of clips in the movie. If, for example, the user deletes the title, iMovie restores all the affected clips.

The *HY4PluginTitleInfo* structure was added with iMovie 3.0.1. It is used only to extend the *HYPluginTitleInfo* structure.

Listing 1-5 HY4PluginTitleInfo structure

```
typedef struct _HY4PluginTitleInfo {
    char    fontName[256];
    // font name; this supersedes the "short fontID" in
    // the HYPluginTitleInfo field which is mostly useless on OS  X
    float   sliderValues[MAX_TITLE_SLIDERS];
} HY4PluginTitleInfo ;//__attribute__ ((aligned (2)));
```

Transition Plug-ins

Transition plug-ins allow the user to select a visual display that eases an otherwise potentially jarring change from one clip to another. Typical transitions are wipes, fades, and dissolves.

A transition is applied between clips; it cannot apply across multiple clips as a title can.

Obtaining User's Direction Choice

The *HYPluginDirection* enumeration defines five values for the user-selected direction:

- kPluginDirectionNone
- kPluginDirectionUp
- kPluginDirectionDown
- kPluginDirectionLeft
- kPluginDirectionRight

For example, if the user clicks the right arrow, iMovie provides kPluginDirectionRight value in *HYPluginTitleInfo.direction*. The plug-in can also supply *HYPluginGetInfo.defaultDirection* to iMovie so it can set the arrow to the desired default when the plug-in is initially chosen.

The *HYPluginTransitionInfo* structure is returned to the plug-in when it makes the GetTransitionInfo callback. The speed member contains the setting for the Speed slider in the transitions pane; its value can be in the range 0.0–1.0. The *ver2* member is defined in the *HYPluginTransitionInfo* structure shown in Listing 1-6 (page 19).

Listing 1-6 HYPluginTransitionInfo structure

```
// HYPluginTransitionInfo
typedef struct _HYPluginTransitionInfo {
    float speed;
    HY2PluginTransitionInfo ver2;
} HYPluginTransitionInfo;
```

The *HY2PluginTransitionInfo* structure is used only to extend *HYPluginTransitionInfo*. It was introduced with iMovie 2.0 to provide the direction member.

Listing 1-7 HY2PluginTransitionInfo structure

```
typedef struct _HY2PluginTransitionInfo {
    HYPluginDirection direction;
} HY2PluginTransitionInfo;
```

Effect Plug-ins

Effects, also know as *filters*, are typically applied to video to add some visual stimulus or to force a particular perspective on the viewer. One of the more popular effects is the water ripple effect, which makes the video appear as if it were projected on the surface of rippling water.

An effect plug-in identifies itself as such by setting the `pluginType` field in the `HYPluginGetInfo` structure to `kPluginFilterType`.

An effect plug-in should apply the effect to an entire clip. iMovie attempts to enforce this restriction by passing only `frameCount` values that correspond to an entire clip, regardless of what the plug-in requests. Moreover, an effect cannot "consume" multiple clips. (The user, however, can apply an effect to multiple clip selections.)

Enabling User Interface Elements

An effect plug-in sets plug-in choice flags (`HYPluginGetInfo.pluginFlags`) to specify user interface elements to be enabled to accept user data, settings, and selections. Plug-in choice flags applicable to the Effects pane are shown in Table 1-10 (page 20).

Table 1-10 Effects pane plug-in choice flag constants

Constant name	Notes
`kFilterChoiceFadeIn`	Value `128L`. Enable fade-in slider.
`kFilterChoiceFadeOut`	Value `256L`. Enable fade-out slider.
`kFilterChoiceAnimated`	Value `512L`. Enable pause slider
`kPluginChoiceFileButton`	Value `2048L`. Enable File… button. (Introduced in version 3 of the API.)
`kPluginChoiceCustomButton`	Value `4096L`. Create and enable a custom button. (Introduced in version 3 of the API.)

The `kFilterChoiceFadeIn` and `kFilterChoiceFadeOut` flags are used to enable or disable the Effect In and Effect Out sliders to gradually ease in (or out) an effect. If the plug-in can't use this functionality, it should dim these sliders.

If the effect applies differently to each frame as it progresses, the plug-in must enable the kFilterChoiceAnimated flag. This feature can force a still clip to be expanded into a clip with enough identical frames such that the effect does not run out of frames before it has completed. If this flag is not set, the effect may be applied to a single frame in a still clip, which is then displayed for the duration of the clip.

Setting the `kPluginChoiceFileButton` flag enables a File... button in the Effects pane. When the user clicks that button, iMovie automatically calls NavServices to choose a file, stores the FSSpec for that file in *`HYPluginFileOpenInfo.chosenFile`*, and passes it back to the plug-in. The plug-in must supply acceptable file types (in the *`typeCount`* and *`hTypes`* fields of). (This feature was introduced in version 3 of the API.)

Setting the `kPluginChoiceCustomButton` flag enables a custom button in the Effects pane. The plug-in is then sent the `kPlugCustomButtonClick` selector when that button is clicked. The plug-in specifies the title of the button in *`HY3PluginGetInfo.buttonTitle`*. (This feature was introduced in version 3 of the API.)