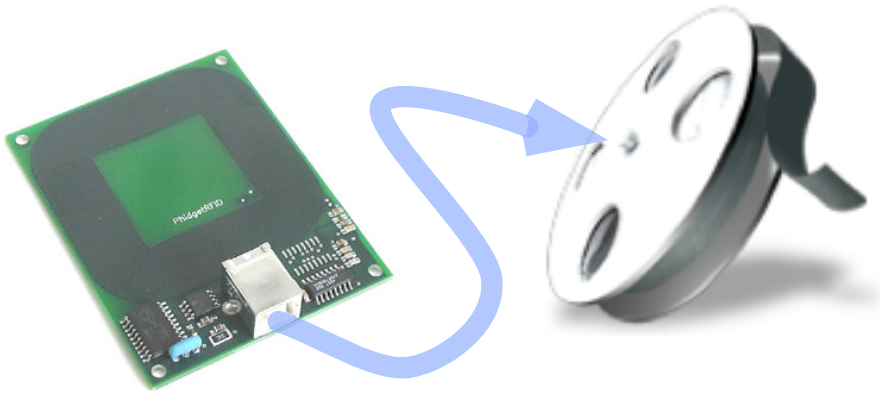

FlashFidgets! User Manual

A guide to configuring the flash player to work the way *you* want with Phidgets

---

Release 2.0a1

Sid Stamm
sstamm@cs.indiana.edu

December 4, 2004

Contents

Chapter 1

Background and Terminology

This chapter contains background information about how the program relates to flash and phidgets, as well as defninitions of terms that might be necessary to know.

1.1 Background Information

describe how the flash player interacts, and its architecture (including the triggers idea)

1.1.1 Obtaining Apple's Developer Tools

If for some reason you wish to modify the source code, or you want to re-compile it, you need to install Apple's Developer Tools. Apple's developer tools are generally packaged with OS X on the installation CD. If for some reason, you don't have them, you can obtain them from Apple's developer site `http://developer.apple.com`. These developer tools contain XCode and all of the other tools necessary for modifying the source code for both the Phidgets Library and the FlashFidgets! program.

1.1.2 Finding a Plist Editor

There's a property list edtior that comes with Apple's developer tools, called (ironically) "Property List Editor" and it is currently my favorite to date. Mostly because it's free. And honestly, editing these plists isn't hard, just ugly if you edit the raw text.

There's also a shareware app out there called PlistEditPro, but it costs $25, and it's not much more advanced than the one Apple distributes.

If you don't want to install developer tools or the shareware plist editor, feel free to edit plists with your favorite text editor. If you do this, don't hold me accountable for making sure you do it right! There's probably a tutorial on the Apple developer website that will show you how to do this.

All the screenshots of editing Property Lists in this manual have been taken while using Apple's Property List Editor.

1.2 Definitions

Array In a plist, an array is an ordered set of elements, usually strings. Each element has an order index, starting with zero.

Data Structure A computer's internal organization of data in a way that makes sense to a programmer. Often, data structures are hierarchical.

Dictionary In a plist, a dictionary is a data structure that stores pairs of data: one key with one value. The entries in a dictionary can be in any order, but each key must be a unique string. The values do not need to be unique and can be any type of data.

Plist Editor A program that presents a user-friendly way to edit Property Lists. You can find one on the web for download (shareware). Apple packages a Plist Editor with the developer tools (installs to `/Developer/Applications/Utilities`).

Property List A configuration file used by many applications in the Mac OS. Property Lists (plists) are stored in XML and can be edited by a plist editor or by hand.

Resource A part of the FlashFidgets! program that is inside the package's contents, hidden from normal view. This way the program and its media can all be encapsulated into one package.

String In computer terms, a series of characters (letters, numbers and symbols) often times surrounded with double-quotation marks to identify it as a string. This is treated as pure data and often has no deeper meaning.

SWF A file created by publishing a flash movie. The FlashFidgets! player can only play files of this type.

Chapter 2

Movies

This chapter will describe what kinds of movies can be played in FlashFidgets! as well as the different methods of opening movies. Movies can be loaded from the filesystem (your folders, hard drive, or desktop) or they can be loaded from a **Bundle Resource.**

You'll want to use movies as resources when you load them by default, fscommand or trigger. All other times (like when you choose "Open..." from the file menu) you will be loading them from the filesystem. This distinction is important, so we'll discuss it in depth.

2.1 Compatability Notes

The whole point of the FlashFidgets! player is to display flash movies. Unfortunately, the current version of Apple's QuickTime (the visual engine for FF) only supports up to Flash version 5.0. This means that when you are writing and designing movies for this application, you need to be sure to publish them in Flash 5.0 compatible SWF files.

2.2 Opening and Playing Movies

Really, this is quite straightforward. Choose the "Open..." menu item from the "File" menu, hold down the Command key and type an O, or click the "open..." button on the toolbar (Figure 2.1).

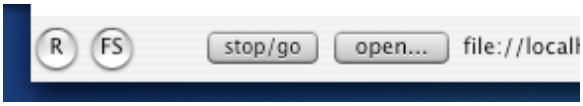

Figure 2.1: The FlashFidgets! toolbar

While we're looking at the toolbar, lets talk about the other buttons. The "R" button reloads the current movie and starts it from the beginning. Use this if you want to refresh a movie for some reason. The "FS" button toggles full-screen mode. You can also hold down Command and type F to do this. Finally, the

"stop/go" button toggles the play mode of the movie: playing or paused. If you pause the movie using this button then click it again, the movie will start where it left off.

2.3 Adding Movie Resources

If you want to add movies to the bundle as resources (for example, setting up triggers) then you need to locate the Resource portion of the FlashFidgets! application.

To reveal this location, you must show the package contents, open the resources folder, and then copy your movies there.

2.3.1 Opening the Bundle

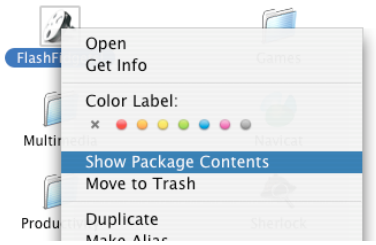

Figure 2.2: Showing the Package Contents

The first thing you have to do is locate the application. It's probably installed in your `/Applications` folder. Find it, and control-click (or right-click) to pop up the context menu (Figure 2.3. From that menu, choose "Show Package Contents." A new window will open in the finder with the contents of the FlashFidgets! package.

Once you've got that open, go into the "Contents" folder and then into the "Resources" folder. This is the folder we're going to work with when we talk about **resources**.

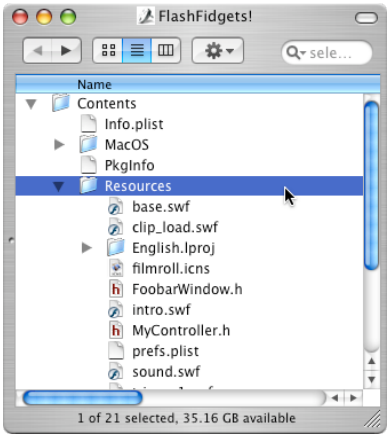

Figure 2.3: The Bundle's Resources Folder

Inspecting `prefs.plist`

Notice that there is a file in the resources folder called "prefs.plist." Why don't we open it up in the plist editor. (Figures in this document will use the one Apple packages with their development tools.) After opening the plist file and expanding the root node, I see that there are a number of useful nuggets in this file (Figure 2.4).

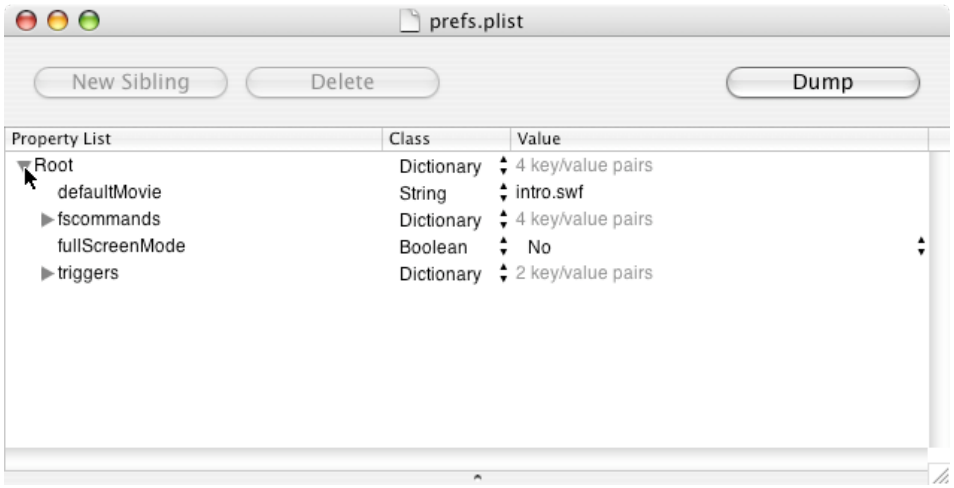

Figure 2.4: Inspecting `prefs.plist`

- `defaultMovie` specifies the *resource* movie to load upon launch. Feel free to change this to any movie in the resources folder. If for some reason the movie doesn't exist (or is blank) the player will simply ignore it.
- `fullScreenMode` specifies whether or not the player should start up in full screen mode.
- `fscommands` is a group of command handlers for interaction with Flash (See Section 3.2).
- `triggers` is a group of trigger sets that define trigger input values and the actions to take (See Chapter 4).

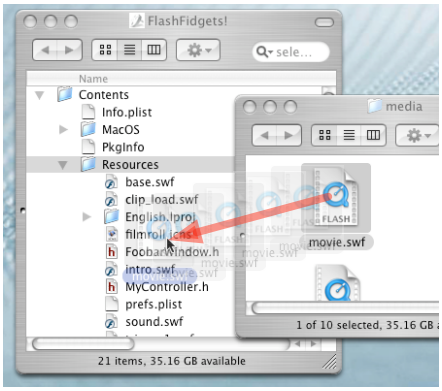

Figure 2.5: Adding Movie Resources with Drag and Drop

Drag & Drop

So how do you put movies into this directory? Well, just make them, then drag and drop from your documents folder (or wherever the SWF files might be) to the resources folder (Figure 2.5).

2.3.2 Accessing these Movies from Flash

Since you cannot access these movies using FlashFidgets! built-in open command (via the button or the menu item), you need some way to open these movie resources.

Of course, you can specify them as triggers or as your default movie. But this is not good enough. Since we can wrap up all these movies into a nice little bundle (as we have done), let's set up a way for a SWF to load another SWF from this resources folder. How? Set up an `fscommand` (see Chapter 3).

Chapter 3

Sending Commands From Flash

If you want to send commands to FlashFidgets! from a SWF movie, you better be familiar (at least a little bit) with ActionScript. You're going to need to attach `fscommand()` calls to events like button-pushes, movie endings, etc.

So right now, do a quick search for `fscommand()` in the ActionScript reference, or at least on google, and find out what it does.

3.1 The `fscommand()` ActionScript Function

`fscommand()` calls are the only way to send information from a SWF movie into FlashFidgets!, so I've created numerous (growing every day) functions you can access via `fscommand` calls. There are only two steps to sending a message from a SWF movie:

1. Add `fscommand( <command>, <args> );` to your event in Flash, then publish it to **Flash 5** format.
2. Update the prefs.plist (see Section 3.2) to contain the new `<command>`.

Say you wanted to add a command in flash that when you pressed a button, you would tell FlashFidgets! to load another movie called "othermov.swf". This is quite easy. Let's call our custom command `openmov`. So we add something to a button push event in Flash that looks like this:

```
fscommand("openmov", "othermov.swf");
```

The next step is to add a command handler in the `prefs.plist` file! Ready for some hacking?

3.2 Adding Command Handlers

How do you handle fscommands? The FlashFidgets! program intercepts all of the fscommands, then looks them up in the `prefs.plist` file to see what to do.

Let's add an fscommand handler. Every command handler has four parts:

1. an **identifier**
2. A **selector**
3. `argv`, the array of default arguments
4. `argc`, the number of default arguments

An **identifier** is the name of the command. If I wanted to create a command called "openmov" then `openmov` would be the identifier.

A **selector** tells FlashFidgets! which internal function to call. These are references to pieces of compiled code, and with each new version it is likely that new selectors will be supported. For now, only a few are supported (See Section 3.3).

3.2.1 Creating a Handler

Let's add a handler. Open the `prefs.plist` file, and expand the "root" dictionary. Expand the "fscommands" dictionary inside the root to expose all of the currently supported fscommands. Click on one (such as "openmovie") and select the "New Sibling" button to create a new sibling (Figure 3.1).

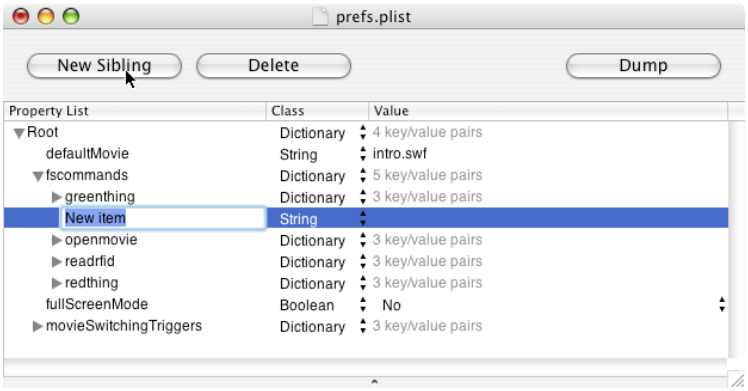

Figure 3.1: Creating a new fscommand Handler

Call this new sibling "openmov" like we used in the example shown in Section 3.1 above. Be sure to set its class to Dictionary in the second column pop-up. (The default is String). Once you've created that, expand "openmov" by clicking the little triangle next to it, and create three children (click the New Child/New Sibling button three times). Set them up to have the following attributes:

identifier	class	value
`argc`	Number	`0`
`argv`	Array	(leave empty)
`selector`	String	`switchToMovieResource`

Once you've entered them, you should have a structure something like shown in Figure 3.2.

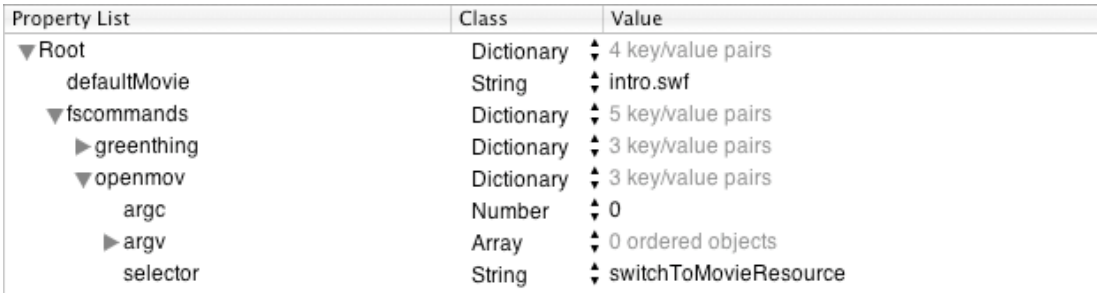

Figure 3.2: The `openmov` handler structure in `prefs.plist`

Now when you want to use this handler, in your action script somewhere, use:

```
fscommand("openmov", "othermov");
```

3.2.2 `args` and `argv`

Why did we create `args` and `argv` if we left them empty? The values specified here in those elements are the **default arguments**: that is, if no arguments are specified by the the `fscommand()` call from ActionScript, these are used instead. (Remember you had a command and an optional argument?) Basically, we'll assume that all SWF movies that want to use will speicify which file to open from the fscommand, not from the plist file.

For handlers like `openmov`, it's useful to provide the arguments directly from flash. But in some cases you won't want to. Say you want a handler `goBackToIntro`. You don't want to specify the name of the intro SWF file in all of your SWFs that call `goBackToIntro`! What if you change the intro's file name? In this case, you want to specify it in the arguments.

To specify a default file, simply change the value of `argc` to 1, and then add a child to `argv` that is a string with the name of the resource movie (such as "target.swf").

3.3 Supported Selectors

Selector	Arguments	Description
`displayDialogAndLog`	(1) message (2) button text	This pops up a little message box to the user and also saves the message to the console log. (You can read this console from the MacOS X Console utility).
`showRFIDTag`	(none)	This pops up a message box that shows the current RFID tag.
`switchToMovie`	(1) movie path	Switches to the movie *file* specified in the argument. Note that the file must be a complete path to the file on the hard drive (e.g. `/Users/sstamm/Desktop/file.swf`)
`switchToMovieResource`	(1) resource	Switches to the movie *resource* specified in the argument. Note that the file must reside in the bundle **resources** directory.
`setMovieTrigger`	(1) trigger set (2) trigger value (3) resource	Sets up a trigger given three things in the arguments: a trigger set name, a trigger value and a movie. (See Chapter 4)

Recall that the arguments can be passed from flash via an `fscommand`, or by specifying their values in the prefs plist.

3.3.1 Sending Multiple Arguments to a Selector

What if you need to send multiple arguments (such as in the case of `displayDialogAndLog`)? Well, there are two ways to do this. First we'll discuss how to specify the arguments in the plist file, then we'll talk about how to pass them in from Flash.

Specifying them in the plist

To specify multiple arguments to a command handler in the `prefs.plist` file, you must modify two things:

1. `argc` must have a value that is the number of arguments you wish to specify (like 2).
2. `argv` must have that number of children. Create the children by expanding argv (click on the triangle) and clicking the "New Child" button.

Be careful when specifying items in `argv`... they must be in the order specified by the table of fscommand handlers above.

Specifying them in the `fscommand`

The other way is to specify the arguments in the fscommand call. Say we wanted to open the file called "filename.swf" that is in the resources folder. We call the `openmov` command that we created before in Section 3.2, by typing (in Flash):

```
fscommand("openmov", "filename.swf");
```

What if we wanted to use `displayDialogAndLog`? That's a little different. Let's assume we've created a command handler called "dialog":

```
fscommand("dialog", "Well hello there cookie!::hi!");
```

Figure 3.3: The Hello Cookie Dialog

Notice how the arguments are separated by a double-colon (`::`)? That's how you specify more than one argument in fscommand calls.

3.3.2 Generalized Structure of an fscommand Entry in the Plist

Each command handler in the `fscommand` dictionary must contain three things: a **selector**, `argc` and `argv`. Argc can be zero, and argv can contain nothing, but the elements must exist. The selector must be one from the table of selectors in Section 3.3.

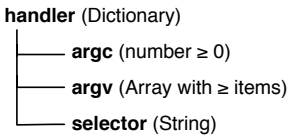

Figure 3.4: General Structure of a Command Handler

3.4 Troubleshooting

Problem	Possible Solutions

Chapter 4

Triggers

Triggers are an optional feature of FlashFidgets! that allow movies to be loaded when certain physical events happen (such as button presses, RFID tag scans, motion, etc. . .)

It is important to realize, although there are a variety of sources (inputs) for triggers, **all they can do is load resource movies**. This is mildly restricting, but it provides a stable flexibility for one input source.

4.1 Overview

The basic model of triggers is that the program sits and watches the source until it changes, and when it changes the new value specifies which movie to load. Triggers are categorized into sets, so that digital inputs *and* RFID tags can perform different actions.

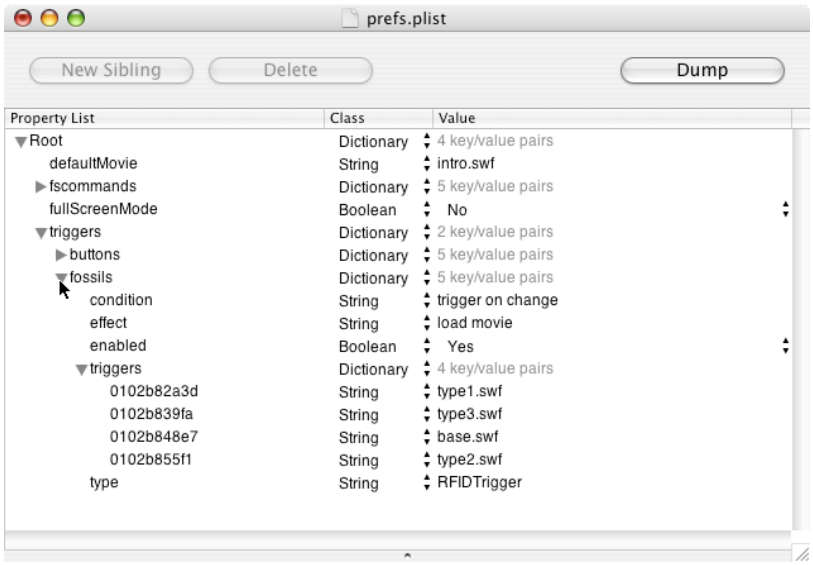

Figure 4.1: The Triggers Portion of the Plist

For example, every time I scan a different RFID card (when PhidgetRFID is the trigger source), FlashFidgets! looks up the value of that card in my and figures out which movie to load.

The first thing you have to do if you want to use a trigger set, is make sure it's enabled on startup. Go into `prefs.plist` and be sure that the `triggers` → `buttons` → `enabled` element is set to `Yes` (Figure 4.1). Once you've enabled them, it is time to specify the source and triggers!

4.2 Specifying Trigger Sets

This is really simple. Each element inside the `triggers` dictionary is a trigger set. Those elements in turn are dictionaries that define the behavior of some sort of trigger: when to activate, what kind of thing to do, whether or not the set is enabled, the list of value–action associations, and the type of input used.

This all sounds scary, but take a look at Figure 4.1 and examine the sub-elements of the `fossils` trigger set. You'll notice that the `fossils` are RFID tag triggers that load a movie when the current tag value changes. Not too bad. Each type of trigger input source has different supported conditions.

4.2.1 Effects

All trigger sets can choose to either load a movie or skip to a specified frame by setting the `effect` to `load movie` or `select frame`.

4.3 Supported Trigger Types

Currently there's only two supported trigger types, `PhidgetRFID` and `Digital Input`.

4.3.1 PhidgetRFID

Obviously this is a Phidget RFID tag reader. Note that this device will **not** trigger twice if you scan the same card twice in a row. It only triggers on **new** values. This is a problem with the HID drivers in OS X, and maybe in the future with a different way to communicating with Phidgets (instead of the provided framework) a more robust method will emerge.

Adding an RFID Trigger

To add an RFID trigger, simply add an item to the `triggers` Dictionary (the one inside a trigger set) with the key being the tag value, and the value being the movie resource or frame number (see Figure 4.1).

How do you find the tag numbers? One way is to create a flash file that displays the current tag using the `showRFIDTag` selector when a button is pushed. The movie called "twobuttons.swf" that is included with the application contains a huge red button that will tell you the current tag. Just place the tag on the reader and then push the button.

Make sure triggers are off when you do this or you might accidentally switch movies!

Conditions

{ `trigger on change` }

The PhidgetRFID type of trigger only supports one condition: `trigger on change`. Unfortunately that's a restriction of the Mac OS X HID system and might get fixed in a future release.

Fudging a Take-Away Trigger

So you know that scanning the same card twice doesn't produce a second trigger, but also removing cards does not cause a trigger. Here's a way to simulate both behaviors.

Place a "base" (or "nothing-is-present") tag underneath the reader at its maximum readable distance. You can test for this distance using triggers — this calibration exercise is left up to you. Now, when you put a tag the reader, it will read whichever is closer: the one you just put there or the "base" tag. When you take it away, the base tag will be read.

To simulate take-away behavior, set up a trigger for the base tag that is some sort of movie asking for input. Instant fudgy behavior!

4.3.2 Digital Input

This uses the Phidget Interface Kit 8:8:8 (USA version). It's important that you use the USA version, because the HID specifications are slightly different.

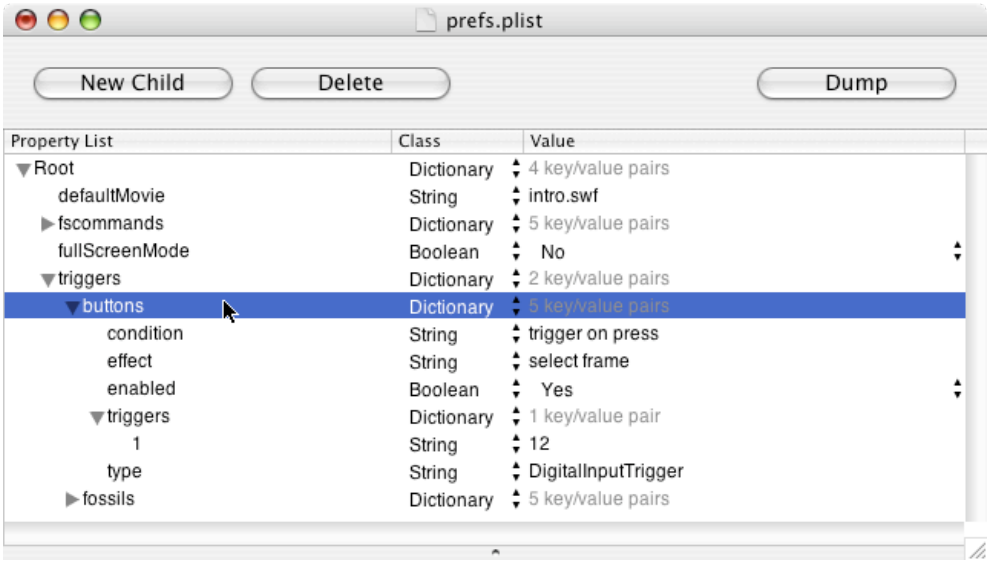

Figure 4.2: The `buttons` Trigger Set

Adding a Digital Input Trigger

To add Digital Input trigger, simply add an item to the `triggers` Dictionary (the one inside a trigger set) with the key being the index (0-8), and the value being the movie resource or frame number (see Figure 4.1).

Conditions

$$\left\{ \begin{array}{l} \texttt{trigger on press} \\ \texttt{trigger on release} \\ \texttt{trigger on change} \end{array} \right\}$$

This type of trigger is a bit more advanced, since it can notice changes from $off \rightarrow on$ and from $on \rightarrow off$, or even both! This means that if you hook up buttons, the interface kit can trigger on button down and up. This means you must specify the trigger condition. This *also* means that you can create two similar trigger sets with one being `trigger on press` and the other being `trigger on release` that do two completely different things—thus providing different behavior for each input when pressed or released.

Chapter 5

Sending Data to SWF Movies

You can't.

5.1 Use Triggers

You can use triggers to load different movies or switch frames based on the input. This is probably the best way to dynamically "send data to SWF movies."

5.2 Selecting Frames

Another thing you could do (that is not fully built in yet) is to skip frames from triggers. If you don't want to have tons of different movies, simply segment out movie clips in the flash file that can only play when their first frame is entered. You can skip around movies by switching frames using triggers quite easily (create a trigger set with the "select frame" condition). The only problem with this is it might be inaccurate — the frame switching feature *has not been heavily tested*. We encourage switching movie files since it is pretty darn fast.

Index