Pippin Launch

version 001

Abstract: This document describes the **Pippin Launch** file used for multiple-application interfaces.

Please send questions and comments via e-mail to pippindev@apple.com.

1 Introduction

The “PippinFinder” Technical Note describes the “replacement” **Finder** file for the Pippin **System Folder**, and that the actual code that drives a multiple application-interface is inside the **Pippin Launch** file. Apple provides an initial interface, a simple button interface, with the buttons changing appearance when they are selected. In summary, Pippin Launch is the file that PippinFinder uses, and this document is describing the ‘PCde’ resource that Apple provides for developers to use. If you wish functionality beyond what `PCde` code implements, you will need to write your own code module. (See the “PippinFinder” Technical Note for details on how to create your own code module.)

☞ If you have a single-title CD and do not want to use a multiple-application interface via **PippinFinder**, you must ensure that the **Pippin Launch** file is removed from your **System Folder** on your Pippin CD. If not removed, you could cause problems with the boot process.

The PCde module allows developers to easily customize the visual appearance of the Pippin startup environment when a Pippin CD is booted. The PCde module will only use the 2Meg provided to it via the **PippinFinder**. This is not of concern since it will all be disposed of before the title is launched.

To distinguish between the Apple provided PCDe module, and potential PCDe modules that may be created by developers, throughout the rest of this document, the Apple provided PCDe will be referenced as “ApplePCDe”.

2 PCde Interface Philosophy

The Launcher presents the user with a single window that contains a background picture and an arbitrary number of buttons.

The background can be a PICT of any size, although it is recommended that the PICT be 640 x 480 pixels in size so it will completely fill Pippin’s screen, much like the Macintosh “desktop”. If the PICT is not big enough to fill the entire screen, the remaining area will be filled with the black.

Each button is a rectangular PICT of any size. There are as many as three PICT resources associated with each button:

- one for the button that is un-highlighted
- one for the button that is highlighted (used when the mouse is within the button rectangle if the resource is present)
- one for the button that is in a “pressed” state (used when the user presses the mouse button when the mouse is within the button rectangle).

Each of these buttons should be the same pixel dimensions. Each, when present, are displayed at the coordinates of the button which are specified separately.

The behavior of the window is that, whenever the mouse is found to be within any of the buttons, the button’s “highlighted” graphic is displayed (if it exists). If a “mouse within” sound exists, it is played as soon as the mouse is detected within the button rectangle. If the mouse leaves the highlighted button, the

button's un-highlighted button graphic is again displayed. No mouse actions other than positioning movement is required by the user to get either graphical and audible feedback.

3 Launcher Resource Overview

The resource fork of the **Pippin Launch** file contains all the data used by the ApplePCde. This includes all graphics and sound data, as well as small descriptive resources which tell the Launcher where to display buttons, how to highlight them, what sounds to play, etc. These resources can be edited with the resource editor *ResEdit 2.1.3* (or higher) utility to create a custom launching environment for Pippin CDs.

There are only two types of resources that need to be created:

- a single "background" resource (i.e., `pbkg`)
- some number of "button" resources (i.e., `pbtn`)

The following sections describe these resource types in more detail. Included in the resource fork of the **Pippin Launch** file on the *Pippin Developer SDK* CD are *ResEdit* "template" resources which simplify the creation of the background and button resources.

3.1 The Background Resource

The background resource, `pbkg`, tells the Launcher which PICT resource are to be the background picture for the launcher's window. It also tells the Launcher which sound, `snd`, should be played when the Launcher first starts up. Only one `pbkg` resource should be present in the Launcher resource fork.

The contents of the `pbkg` resource are as follows:

Background PICT id - This is the resource ID of a PICT resource that is the background picture for the Launcher window. The dimensions of the PICT are used to set the size of the Launcher window.

Background PICT handle - This is used at runtime to hold a handle to the background picture and should not be filled in by the user. *Reserved*

Startup snd id - This is the resource ID of a `snd` resource. The sound, if specified, is played when the launcher starts up and can be used to play introductory music or instructions. Set this to zero if you don't want to use a startup sound.

3.2 Button Resources

There can be any number of button resources present (i.e., any number of the resource `pbtn`). Each describes where a button is placed in the window, whether it is a hot highlighted or un-highlighted graphic, whether it is a "pressed" graphic, and what sounds should be played for the "mouse entering the button" and "mouse pressing the button" events.

The contents of the `pbtn` resource are as follows:

Button rectangle. - This is used to specify the top left coordinates of the position of all button art for the button. The bottom and right members of the rectangle are ignored by the Launcher and are calculated using the size of the button PICT. Only the top and left fields are necessary. *Required*

Button PICT id - This is the resource ID of the PICT for the un-highlighted, or normal state, of the button. *Required*

Button PICT handle - This is used at runtime and should not be filled in by the user. *Reserved*

Highlighted button PICT id - This is the resource ID of the PICT for the highlighted state of the button. This PICT, if specified, is displayed whenever the mouse is determined to be within the button rectangle. *Optional*

Highlighted button PICT handle - This is used at runtime and should not be filled in by the user. *Reserved*

Button pressed PICT id - This is the resource ID of the PICT for the pressed state of the button. This PICT is displayed when the mouse button is pressed and is within the button rectangle. *Optional*

Button pressed PICT handle - This is used at runtime and should not be filled in by the user. *Reserved*

Mouse within snd id - This is the resource ID of the sound (‘snd’) resource that is played whenever the mouse enters the button rectangle. *Optional*

Mouse within snd handle - This is used at runtime and should not be filled in by the user. *Reserved*

Launch object type - The value of this field specifies what the ApplePCde should do with the file specified by the “App or doc filename” field when the user presses the mouse button while the mouse is within the button rectangle. *Required*

Values for this field are as follows:

0	no action; ignore the “launch object” file
1	launch object is an application file
2	launch object is a document file

App or doc filename - This field contains the full HFS pathname of an application or document and generally is the action that is performed when the user presses the button. Applications and documents are simply launched.

If this field is left empty, ApplePCde will take no action when the user presses the button.

It is crucial that you get this field name correct, any errors in the path name could result in a hang of the OS or a crash.

4 Customizing ApplePCde

This section describes a step-by-step process for creating custom ApplePCde resources.

1. Make a copy of the **Pippin Launch** file so that you do not modify your original. It will be easier to “start over” or begin a new project if you keep an unmodified copy of **Pippin Launch** on hand.
2. Prepare your Pippin Launch art and sound files. This may be done using any tools which can create Macintosh PICT and snd data. Many Macintosh graphics programs can save graphics in PICT file format. However, the ApplePCde requires all data to reside in its “resource fork”.

Some programs, such as *Adobe Photoshop*, permit graphics to be saved directly as a PICT resource file. This can simplify your work. Otherwise, you can create graphics and use the Clipboard to copy and paste them directly into the **Pippin Launch** resource fork using *ResEdit*.

Sound resource data can be created using the Macintosh microphone or with A/V sound input capabilities. Most likely, for high quality audio, the microphone will not be used. However, it can be useful to "prototype" your sounds with the microphone using the resulting low quality sound as place holders for your final data.

4.1 Creating Launcher Art

The first step is to choose your background art. The graphic should be a full screen image (640 x 480 pixels) so as to cover the Pippin TV screen. If you have a fully rendered scene, you may only need to cut rectangles out of the background (like a cookie cutter) in order to form your un-highlighted button art. However, button art can be anything you like, even something totally unrelated to the background.

Once you have chosen your background picture, save it as a PICT resource so that you can import it later using *ResEdit*. Alternatively, you can open the Pippin Launch file using *ResEdit*, and paste the PICT directly into the **Pippin Launch**'s resources. If you do not have a graphics program that supports the creation of PICT resources, you will need to import your art this way.

4.2 Assigning PICT Resource IDs

When you paste a PICT into the **Pippin Launch** resource file, *ResEdit* assigns a number, called a resource ID, to the data. This number gives the picture a unique name amongst all PICTs in the file. You should assign a resource ID to each picture to help you recall which picture is which. You might choose to use a certain range of numbers for each of the button pictures so that it is easy for you to know which pictures are related to each other by their ID. For example, you might use the IDs 150, 151, and 152 for a button, its highlighted state, and its pressed state, respectively. You might use 160, 161, and 162 for another button. In this example, it's easy to tell which PICTs are associated with each other, and what each of them is by looking at the resource ID. While this methodology can help you keep things organized, it is not required.

4.3 Button Rectangles

Whether you decide to use the "cookie cutter" method mentioned earlier for making buttons from a fully rendered scene, or some other method altogether, you will need to gather some precise information about the placement of the button art work. If you are cutting buttons out of a background, you will want to use a graphics program that can tell you the XY position of the rectangular patch you copy from the background. This information must be entered into the button rectangle field of the `pbtn` resource, each of which is used to store button information. This information must be transcribed exactly or the button will not be drawn in the correct position.

4.3.1 Making Highlighted Buttons: An Example

Once you have cut out buttons from a background, or other source, you may wish to make a graphic for the button's highlighted state. This will be displayed whenever the user's mouse pointer is within the

rectangle of the button. This is a form of “hot highlighting” that can be used to indicate that a button exists in an area of the screen. This art is optional. If it is not present, the button will not appear different when the mouse is “over” it.

One technique for making highlighting button art is to begin with a copy of the original button art. Using *Adobe Photoshop*, for example, you can form a selection area around the edges of whatever graphic element is prominently considered as the button. Invert the selection, and then use an airbrush tool to make a colored halo around the art. This technique can be used to create the appearance of highlights of arbitrary shape and color.

Similar techniques can be used to make “button pressed” art. Keep in mind, though, that the art for the button, its highlighted state, and its pressed state do not have to be related to each other. In fact, a completely different graphic for each state may be appropriate, depending on the “look” you are trying to create. For example, you might use a person’s face as the button, the same face with raised eyebrows for a highlighted state, and the same face with an open eyed, open mouthed expression for the pressed state.

4.4 Using Sound

Sounds can be used to augment the Launcher’s behavior. You can use a “startup sound” that is played when the Launcher first starts, perhaps playing a musical theme or an introduction to your product: “Welcome to XYZZY...”. Sounds can also be played whenever a user puts the mouse pointer over a button. This can be useful as a audible prompt to “try pressing me”, for example, or as something instructional about what will happen if the button is pressed.

Launcher sounds are loaded into memory, so whatever you decide to use must fit into Pippin’s memory. You can economize, for example, by using the same sound for all button presses.

To use a sound, simply “paste” it into the Launcher resource file using *ResEdit*, and assign the sound’s resource ID to one of a button’s functions by typing in the sound’s resource ID into one of the “snd ID” slots of a button: mouse within, button pressed.

4.5 Launch Object

A launch object determines what happens when the user presses a given button. There are three possible actions:

- do nothing
- launch an application
- open a document

The action of “do nothing” seems likes it is not very useful. However, you can use these types of buttons to play audible instructions by simply associating the appropriate sound resource with the button press. If the user presses the button, some instructions are played out loud, but no other action is taken.

The next two actions, launching an application and opening a document, are self-evident as to what they do. You would use launch a document to open authored files, such as Hypercard stacks. For a stand-alone application, you would simply launch an application. Of course, if there is only one application on your CD, you would probably just want to launch it directly as described in the “Pippin Startup Process” Technical Note and it is not necessarily for you to use **PippinFinder** or the **Pippin Launch** mechanism.

4.6 Customizing PCde Code

While the source code is currently not provided for the ApplePCde resource, it is simple enough for a developer to write his own PCde in C or Pascal (Actually in any language you want as long as it is able to support Pascal calling conventions at its entry point).

4.7 Memory Utilization

When **PippinFinder** calls to the existing PCde module, it has created a 2Meg handle in temporary, initialized a heap into it, and set the current zone to that newly allocated memory. Since control is maintained by the code module until a launch selection is made, the heap remains valid till control is passed back to **PippinFinder**. When **PippinFinder** regains control, the heap is disposed and the specified FSSpec is launched.

4.8 Use of Temporary Memory

When launching applications, Macintosh system software creates a heap zone in the “Multifinder heap” based on the ‘SIZE’ resource information (familiar to users as the Finder’s “Get Info” information about an application icon). All resource and other memory allocation takes place within that heap zone unless the application software takes explicit steps to manage memory in other ways. Most applications are designed to simply use their own application heap.

Beginning with System 7.0, application programming interfaces for the use of so called “temporary memory” were made public. Temporary memory is simply memory allocated from the Multifinder heap, the same heap used for application heap allocation. As such, programmers were directed to not use temporary memory for extended periods of time because it could cause the Multifinder heap to become fragmented and, thus, prevent applications from launching due to the lack of appropriate contiguous free space.

Since Pippin is intended primarily as an multimedia playback appliance, however, PCde developers are encouraged to use temporary memory if their memory needs exceed the 2Meg heap provided to them. Since **PippinFinder** does not know about the temporary memory being used by the PCde module, the module must make the effort to release all used memory in this area before returning code to **PippinFinder**.