Upgrader 1.1.1 & ModifierTool 1.1.1b3
Technical Guide

®

Document Version 1.2 - January 12, 1998

Contents

Chapter 3 Writing Upgrader Plug-ins

Introduction

The Upgrader application provides a programming environment for creating assistant-like programs to guide users through the on-screen panels necessary to complete an installation or setup task. Each **panel** is a window that prompts the user to perform one step of the task. The panels are implemented by **plug-ins**, which are individual files containing code written by a developer that can be assembled together into a single user experience. To control the order of the plug-ins and provide the information required by each plug-in, the developer creates a **data file**. The Upgrader application, plug-in files and data file are collected together by a developer into a group of files referred to as an **Upgrader-based program.** One example of a shipping Upgrader-based program is the Install Mac OS 8 program used to install Apple's Mac OS 8 system software.

The companion application, **ModifierTool,** allows modifications by clients of an existing data file or creation of a new one by a developer. A **client** might be an IS manager or other developer who leverages from an existing Upgrader-based program, and possibly other plug-ins, to address a unique scenario, such as tailoring the Install Mac OS 8 program for a company's internal needs.

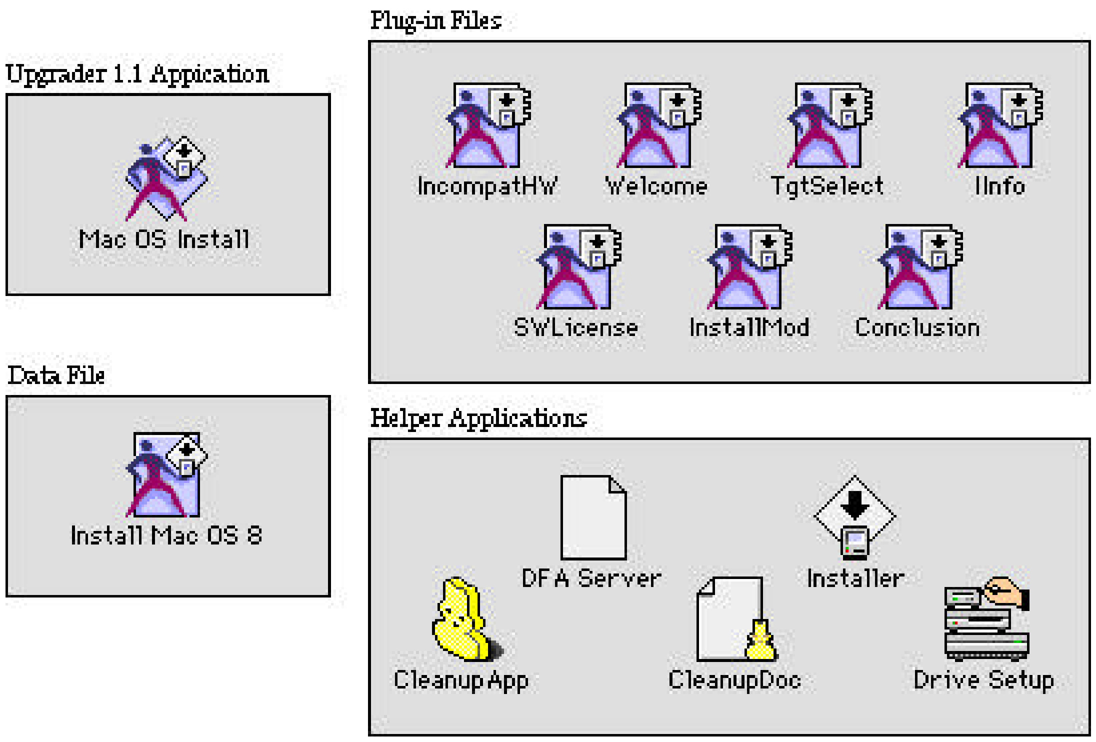
We begin our discussion with a brief overview of the Upgrader and ModifierTool applications, then present a road map to help readers decide which parts of this documentation are pertinent to their needs.

About Upgrader

Programs created using the Upgrader application as their foundation will normally be a collection of many files, all working together to present a single, cohesive user experience. This type of modularity enables new programs to be created more quickly and existing programs, such as the Install Mac OS 8 program, to more easily be modified for future requirements.

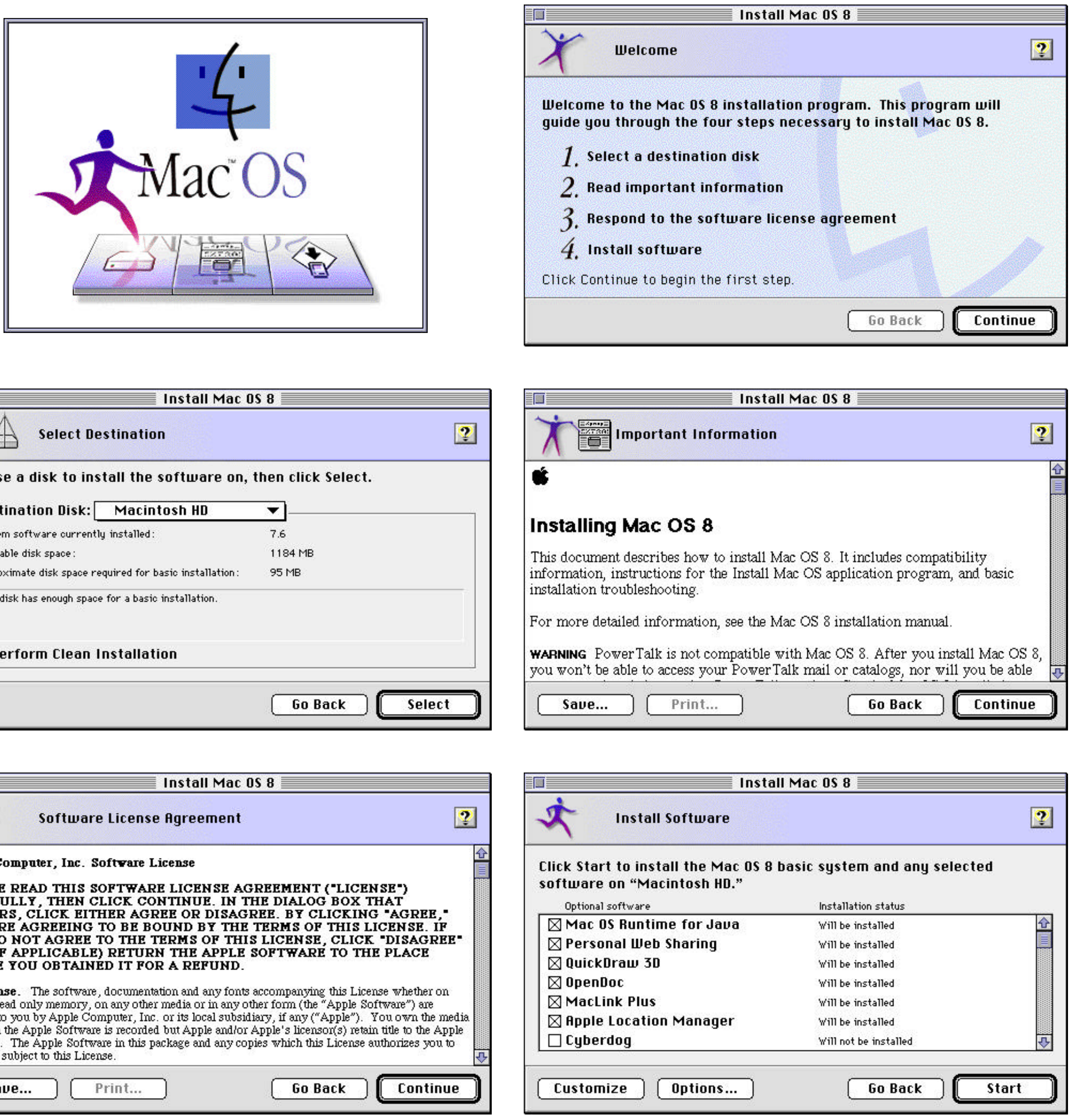
Figure 1-1 shows the files that implement the Install Mac OS 8 program. Not only does the modularity of an Upgrader-based program make later modifications easier, but it also enables complex disk layouts, such as those required on floppies or multiple CD-ROMs.

Figure 1-1 Files implementing the Install Mac OS 8 program



From this collection of files the users see a simple series of panels, which guides them through a series of steps. Figure 1-2 shows the panels a user sees when upgrading to Mac OS 8.

Figure 1-2 The Install Mac OS 8 program panels



Welcome to the Mac OS 8 installation program. This program will guide you through the four steps necessary to install Mac OS 8.

1. Select a destination disk
2. Read important information
3. Respond to the software license agreement
4. Install software

Click Continue to begin the first step.

Choose a disk to install the software on, then click Select.

Destination Disk: Macintosh HD

System software currently installed:	7.6
Available disk space:	1184 MB
Approximate disk space required for basic installation:	95 MB

This disk has enough space for a basic installation.

☐ Perform Clean Installation

Installing Mac OS 8

This document describes how to install Mac OS 8. It includes compatibility information, instructions for the Install Mac OS application program, and basic installation troubleshooting.

For more detailed information, see the Mac OS 8 installation manual.

WARNING PowerTalk is not compatible with Mac OS 8. After you install Mac OS 8, you won't be able to access your PowerTalk mail or catalogs, nor will you be able

Apple Computer, Inc. Software License

PLEASE READ THIS SOFTWARE LICENSE AGREEMENT ("LICENSE") CAREFULLY, THEN CLICK CONTINUE. IN THE DIALOG BOX THAT APPEARS, CLICK EITHER AGREE OR DISAGREE. BY CLICKING "AGREE," YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, CLICK "DISAGREE" AND (IF APPLICABLE) RETURN THE APPLE SOFTWARE TO THE PLACE WHERE YOU OBTAINED IT FOR A REFUND.

1. License. The software, documentation and any fonts accompanying this License whether on disk, in read only memory, on any other media or in any other form (the "Apple Software") are licensed to you by Apple Computer, Inc. or its local subsidiary, if any ("Apple"). You own the media on which the Apple Software is recorded but Apple and/or Apple's licensor(s) retain title to the Apple Software. The Apple Software in this package and any copies which this License authorizes you to make are subject to this License.

Click Start to install the Mac OS 8 basic system and any selected software on "Macintosh HD."

Optional software	Installation status
☒ Mac OS Runtime for Java	Will be installed
☒ Personal Web Sharing	Will be installed
☒ QuickDraw 3D	Will be installed
☒ OpenDoc	Will be installed
☒ MacLink Plus	Will be installed
☒ Apple Location Manager	Will be installed
☐ Cyberdog	Will not be installed

For the most basic Upgrader-based program, you'll need to understand the purpose of three file types: the Upgrader application file, plug-in files, and the data file.

Upgrader Application File

The Upgrader application file provides a set of services for implementing panels. Plug-ins use the **Upgrader API** to help handle user interaction by using provided routines to receive user events, control panel navigation, display text and dialogs, and more. The file also contains resources that are needed by all the plug-ins, such as 'DLOG' resources, help icon 'PICT's and other common resources.

The user will double-click the Upgrader application to start an Upgrader-based program. The Upgrader application file is often referred to as the "shell" because it manages the relationship between all plug-in files and the data file.

System Requirements

The Upgrader application is designed to run on 68020-based and newer machines running system software version 7.0 and later. Apple’s primary testing has been performed on 68040 and PowerPC machines running 7.1 and later, so developers wishing to support earlier versions are encouraged to perform additional testing on older configurations.

Partition Size

The Upgrader application is shipped with a partition of 750K, but some plug-ins may required a larger partition to work correctly. Use the following equation to help you determine the appropriate partition size:

Partition size = plug-in memory requirements + size of preload resources + 300K

Calculate the “plug-in memory requirements” value by choosing the plug-in with the highest memory requirements. The plug-in documentation should provide the memory requirements of the plug-in. If none is provided, use 200K.

Calculate the “size of preload resources” by using the Data File Memory Requirements value displayed in the main data file window of ModifierTool.

Upgrader Plug-in Files

A plug-in implements the actual code to draw the panels and interact with the user. Each plug-in file usually manages one or more panels, but it may also operate without any user interaction. Since only one plug-in runs at a time, it controls the Upgrader-based program until it relinquishes control to the next, or previous, plug-in.

A plug-in file normally contains a code resource and various resources needed by that plug-in. The plug-in might also depend on information stored outside the plug-in file, such as shared resources stored in the Upgrader application file, client-controlled resources in the data file, and possibly other helper applications that are run either in the background or foreground to support the plug-in.

Upgrader Data File

The Upgrader data file contains information read by the shell and plug-ins to present the panels in the correct order and display the panel contents. The developer of a new Upgrader-based program creates a data file containing the information needed by the plug-in contained in the program. After the release of the developer’s Upgrader-based program, clients (such as an IS manager) can augment the information in the data file to adapt the program to their needs.

Each plug-in usually requires one “preference” resource in the data file, which may reference many other resources contained in the data file. To help manipulate these many resources, editing tools are normally provided by the developer.

Locating the Data File

When the user double-clicks the Upgrader application icon, the Upgrader searches for a single data file in the same folder as the application. An alert allows the user to choose a data file if none or more than one is found.

The developer should always lay out the CD-ROM or floppy to encourage the user to double-click the Upgrader application, instead of the data file. This lessens the chance of another — perhaps incompatible — version of the Upgrader application being launched by the Finder. We suggest hiding the data file outside of the disk or folder’s window border.

Choosing an Editing Tool

To create and / or modify the resources in the data file, developers and their clients have three options: use a Rez complier in concert with the UpgraderTypes.r file (in addition to a resource definition file provided by each plug-in developer), use Resorcerer and our template file, or use the ModifierTool application. All the necessary files are included on the SDK.

Each editing method has its own benefits, but Apple recommends using ModifierTool, since this application makes it easier to maintain the integrity of complex data files. This document will focus on the ModifierTool approach to editing data files, but the contents of the UpgraderTypes.r file is detailed in the “Editing Upgrader Plug-ins” chapter and in the section “Upgrader API Reference” at the end of the “Writing Upgrader Plug-ins” chapter.

Apple encourages developers who write their own plug-ins to also write a ModifierTool editor so clients, present and future, can easily modify the plug-in’s functionally.

About ModifierTool

Most developers, and their clients, will use ModifierTool to create and change the information in their Upgrader data file. The architecture of ModifierTool application and its editor files has been designed to match the modular design of Upgrader and its plug-in files. ModifierTool contains a built-in editor for the information the Upgrader application uses to load and run each plug-in in the desired order.

Using This Document

Since not all readers of this document will have the same goals, we present several strategies based on how you may need to interact with the Upgrader and ModifierTool applications.

If you just need to make changes to the Install Mac OS 8 program or a program created by someone else, read:

- the “Editing Upgrader Plug-ins” chapter.

If you need to create a new Upgrader-based program using existing plug-ins, read:

- the first section of the Editing Upgrader Plug-ins chapter
- any documentation that came with the plug-ins you wish to use.

If you need to write a new plug-in, read:

- the chapter “Writing Upgrader Plug-ins”
- the chapter “Writing ModifierTool Editors”, if you want to make it easy for clients to modify your plug-in.

Editing Upgrader Plug-ins

This chapter describes how to edit an Upgrader data file using the ModifierTool application and provides details about each plug-in shipped with the Install Mac OS 8 program. This chapter describes all the resources that make up an Upgrader data file.

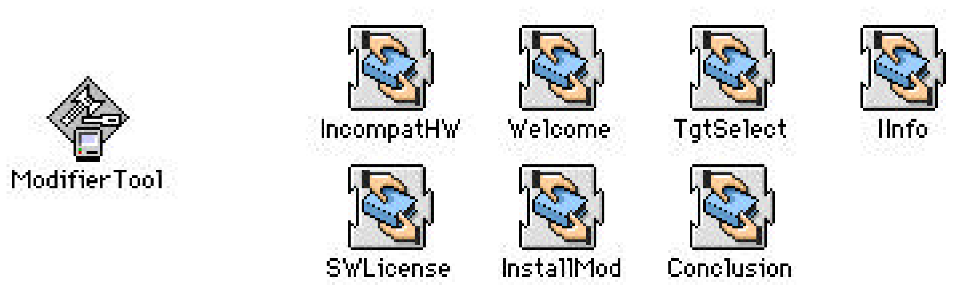
Using ModifierTool

To use ModifierTool, you must have at least the following minimum configuration:

- PowerPC-based computer
- Mac OS 7.1.2 or later
- 2 MB of available memory

Before you begin, make sure all the editors you'll need are in the same folder as the ModifierTool application. Figure 2-1 shows the typical layout of the ModifierTool application and its editor files.

Figure 2-1 ModifierTool application and editor files



We'll use the Install Mac OS 8 program in most of our examples as we take the reader through the editing tasks necessary to create and modify the Upgrader application-owned resources in the data file.

Creating and Opening a Data File

To create a new data file, launch the ModifierTool application and choose New from the File menu. You will be you asked to name the data file and choose a location to save it. Once the ModifierTool creates the sequence resource and default settings, the main data file window will open (see Figure 2-2), ready for you to begin defining the plug-ins that will make up your Upgrader-based program.

If you need to edit an existing data file, launch the ModifierTool application and choose Open from the File menu then select the desired data file. You may also drop the data file icon on the ModifierTool application icon to open the file.

NOTE

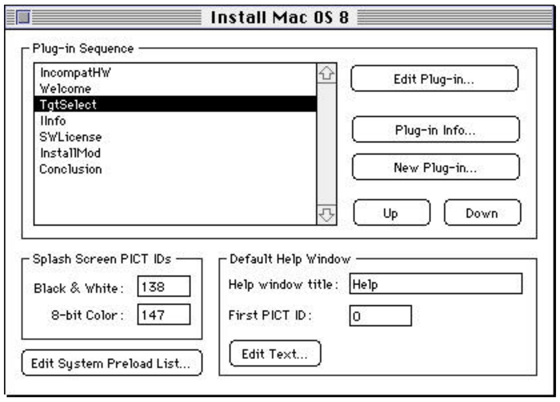
ModifierTool cannot open a data file that is on locked media or is already open for modification by another application.

Since one only data file can be open at one time, you will need to close the currently open data file by choosing Close from the File menu before choosing New or Open.

Using the Main Data File Window

The main data file window provides the user with access to the individual plug-in editors, as well as Upgrader application preferences that control attributes such as the order of the plug-ins, the splash screen picture, the default help text, and the list of System resources to be preloaded when running on ejectable media. This window is always present as long as the data file is open. If you close this window, the data file will be closed. If changes were made in this window (including adding or removing plug-ins), you will be asked if you wish to save your changes before closing the window or quitting the ModifierTool application.

Figure 2-2 Main data file window


Install Mac OS 8

Plug-in Sequence
- IncompatHW
- Welcome
- TgtSelect
- IInfo
- SWLicense
- InstallMod
- Conclusion

Splash Screen PICT IDs
Black & White: 138
8-bit Color: 147

Default Help Window
Help window title: Help
First PICT ID: 0

The items in the main data file window are explained in detail below.

Plug-in Sequence:

Plug-in List	Shows the current list of plug-ins in the default order they will be presented to the user. Click on the name of a plug-in to select it.
Edit Plug-in…	Opens the editing window for the selected plug-in. This window is implemented by an editor file with the same name as the plug-in file and in the same folder as the ModifierTool application. If an editor cannot be found, the plug-in info window is opened instead.
	Double-clicking on a plug-in name is a shortcut for opening a plug-in’s editor. If no editor exists, the plug-in info window is

	opened instead.
Plug-in Info…	Shows the plug-in info window for the selected plug-in so attributes of the plug-in can be changed. See the section “Using the Plug-in Info Window” for more information about this window.
New Plug-in…	Shows an empty plug-in info window. If the user clicks OK in the plug-in info window, the new plug-in entry will added to the plug-in list. If a plug-in is selected in the list when clicking New Plug-in, the new plug-in will be inserted before the selected plug-in; otherwise, it will be added to the end of the list.
Up	Moves the selected plug-in one entry closer to the beginning of the plug-in list.
Down	Moves the selected plug-in one entry closer to the end of the plug-in list.
Default Help Window:	
Help Window Title	The window title of the help window if the plug-in does not set the help window’s name upon entry.
First PICT ID	The 'PICT' resource ID of the first picture to be displayed in the help text. See the section “Using the text editor window” in this chapter for more information about adding pictures to your text.
Edit Text…	Opens the text editor window, so the text can be changed. See the section “Using the text editor window” for more information about editing text.

NOTE

Make sure to save the data file if you clicked Remove in the file reference window.

Splash Screen PICT IDs:	
Black & White	Resource ID for your black and white splash screen PICT resource. This is a B&W version of the splash screen for monitors showing less than 256 colors. Enter 0 if you do not wish to provide a separate B&W picture.
8-bit Color	Resource ID for your 8-bit color splash screen PICT resource. This is a color version of the splash screen for monitors showing 256 colors or more.

The default splash screen provided with Mac OS 8 data file can be replaced, if necessary. Two 'PICT' resources should be created, a color version and a B&W version. The dimension of your splash screen picture should be no larger than 320 pixels in height by 500 pixels in width. The Upgrader application will automatically resize the window based on the size of the picture.

Once you have created your two new 'PICT' resources follow these instructions to replace the default splash screen resources.

Use a resource editing program, such as ResEdit or Resorcerer, to open the data file.

1. Remove the existing splash screen 'PICT' resources of IDs 138 and 147.
2. Paste your new color splash screen 'PICT' resource into the data file and renumber its ID to 147.
3. Paste your new B&W splash screen 'PICT' resource into the data file and renumber its ID to 138.
4. Quit the resource editing program and open the Upgrader application to verify that your splash screen is displayed.

If you choose to use different IDs for your splash screen 'PICT' resources make sure to change the IDs in this window.

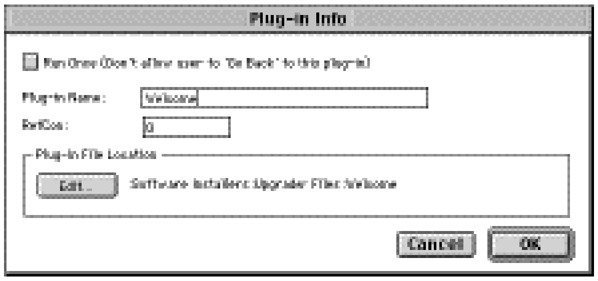
Other items:

Edit System Preload List…	Opens the resource preload editor window so the list of System resources to be preloaded at launch time can be modified. You'll only need to worry about this list if you ship a floppy disk set which allows the user to boot from the first floppy disk. If you do, then you may need to add System resources to this list that plug-ins or helper applications require during their operation. If you fail to add these additional resources to this list, extra disk swaps back to the System floppy disk will be required, which can degrade the user experience of your Upgrader-based program.

Using the Plug-in Info Window

The Upgrader needs to know basic information about each plug-in, such as the plug-in's name and where to find the plug-in file. The plug-in info window allows the user to edit this information (see Figure 2-3).

Figure 2-3 Plug-in info window



Plug-in Info

☐ Run Once (Don't allow user to 'Go Back' to this plug-in)

Plug-in Name: Welcome

RefCon: 0

Plug-in File Location

Edit… Software Installers:Upgrader Files:Welcome

Cancel OK

The items in the plug-in info window are explained in detail below.

Run Once checkbox	Prevents the user from going back to this plug-in. The Go Back button in the Upgrader-based program will automatically be dimmed if the preceding plug-ins are set to run once.
	The Environmental Filter plug-in included with the Install Mac OS 8 program uses this feature, since the user's machine only needs to be checked once during the launch of the program.
Plug-in Name	Any name that describes the plug-in.

Refcon	A 4-byte value passed to the plug-in upon initialization. Most plug-ins will want to use the low-word of this value to store the resource ID of its preference resource. This allows for a single plug-in file to be used for multiple plug-ins, each with a different resource ID.
Cancel	Ignores any changes made in the plug-in info window.
OK	Keeps any changes made in the plug-in info window.

Plug-in Location:

Edit…	Opens the file reference editing window to edit the location of the plug-in file. ModifierTool uses the name of the selected plug-in file as it appears on the disk to locate the proper editor.

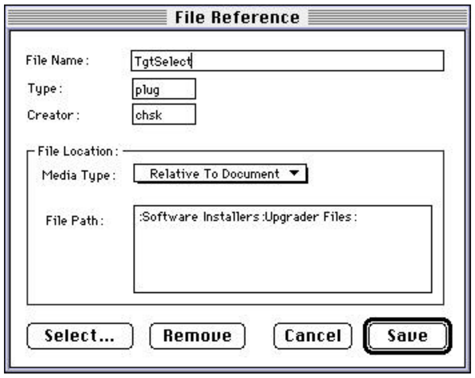
NOTE

Make sure to click OK and then save the data file if you clicked Remove in the file reference window.

Using the File Reference Window

The modular nature of an Upgrader-based program requires references to numerous files. Windows that contain fields referencing a file usually have an Edit button which presents the file reference window (see Figure 2-4). From here the user can fill in the details required to locate the file during the operation of the Upgrader-based program.

Figure 2-4 File reference window


File Reference

File Name: TgtSelect

Type: plug

Creator: chsk

File Location:

Media Type: Relative To Document

File Path: :Software Installers:Upgrader Files:

The items in the file reference window are explained in detail below.

File Name	Name of the file. Limited to 31 characters.
Type	Four character “type” of the file. Leave this field empty (no characters, not even spaces) to cause Upgrader to ignore the file’s type when locating the file.
Creator	Four character “creator” of the file. Leave this field empty (no characters, not even spaces) to cause Upgrader to ignore the file’s creator when locating the file.

Select…	Opens the standard Get File dialog for choosing a file. After clicking OK in the Get File dialog, ModifierTool enters information about the chosen file, overwriting the current contents of the fields. Make sure to select the desired media from the Media Type pop-up menu before clicking the Select button to have ModifierTool generate the appropriate path. To generate a correct relative path, place the data file you are editing in its actual location on your source disk before clicking Select.
File Location:	
Media Type	The setting of this pop-up menu tells the Upgrader application how to find the specified file. If media type is set to "Relative to Document", then the Upgrader-based program expects a relative path from the folder that contains the Upgrader application. If media type is set to "Full Path to Floppy" or "Full Path to CD-ROM" then the path is assumed to be a full path beginning with the name of the floppy or CD-ROM disk. To support creating a net install from a set of CD-ROM or floppy disks, the Upgrader-based program will first look in the folder containing the data file to find a folder with the same name as the CD-ROM of floppy disk before asking the user to insert the disk. Changing the Media Type pop-up menu does not automatically modify the contents of the Path field.
Path	This should be a relative path from the folder containing the data file or a full path beginning with the name of the floppy or CD-ROM disk. Use of a full or relative path depends on the media type. See the "Media Type" field description for more information. If the path is relative, then it should start with a colon. A full path should start with the name of the floppy of CD-ROM disk, instead of a colon.

Using the Text Editor Window

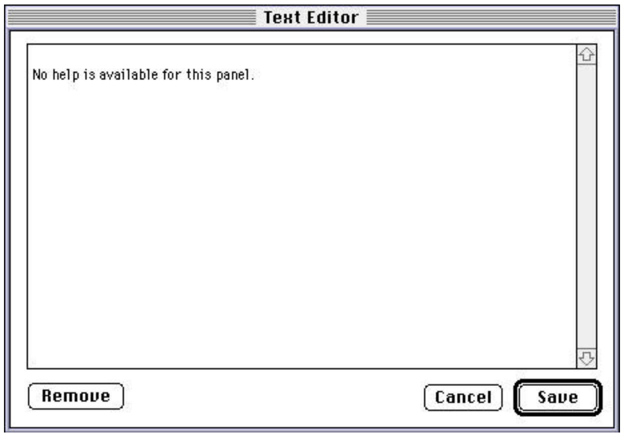
Text stored in the data file is updated using the text editor window.

NOTE

Since the text editor window has no controls for changing the font, font size, font style or any other text-related attribute, we suggest that you store your text in a separate document using your favorite word processor. To change the text, copy the text from your word processor, then paste the text into the text editor window.

The text display engine used by Upgrader supports pictures embedded within the text which is compatible with SimpleText documents. The text display engine uses a special character (usually Option-space, but it can be set by the plug-in) to define where the top of the picture should be drawn. See *Appendix A* for more information on embedding pictures in text displayed in an Upgrader panel.

Figure 2-5 Text editor window



No help is available for this panel.

The items in the text editor window are explained in detail below.

Remove	Deletes the text resource (type: 'TEXT' & 'styl') from the data file and closes the text editor window.

NOTE

Make sure to click Save in the window from which you opened the text editor window to correctly update the data file.

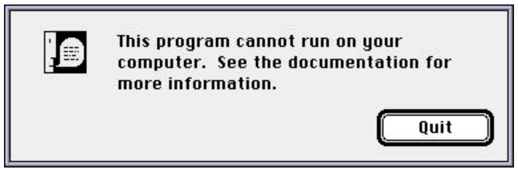
Cancel	Closes the text editor window without changing the text resource.
Save	Updates the text resource in the data file with the contents of the window.

Editing the Environment Filter Plug-in

The environment filter plug-in alerts the user if the computer model is not supported or the version of system software running is too old. For either case, the user is presented with the alert in Figure 2-6.

Plug-in file name: IncompatHW

Figure 2-6 Incompatible environment alert



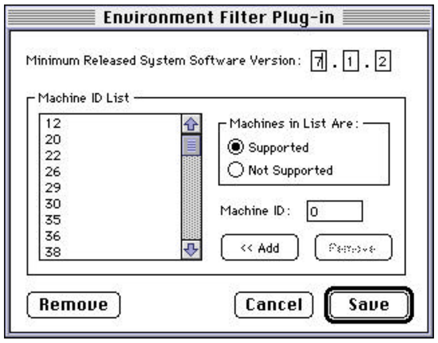
This program cannot run on your computer. See the documentation for more information.

No alert is presented if the environment is sufficient.

Using the Environment Filter Plug-in Editor

The environment filter editor is shown in Figure 2-7. This editor allows the developer or client to change which computer models and system software versions the Upgrader-based program will run on.

Figure 2-7 Environmental Filter plug-in editor



Minimum Released System Software Version: 7 . 1 . 2

Machine ID List: 12, 20, 22, 26, 29, 30, 35, 36, 38

Machines in List Are: [✓ Supported] Not Supported

Machine ID: 0

The items in the environment filter plug-in editor window are explained in detail below.

Minimum Released System…	Enter the minimum Mac OS system version on which your Upgrader-based program should be run. Since the Upgrader application requires at least system 7.0, enter a version of 7.0.0 or higher.
Machine ('mach') ID List:	This is a list of the computer models supported / not supported (see “Machines in List Are:” below) by your Upgrader-based program. Each computer model is identified by its gestalt 'mach' ID. You can add IDs to or remove IDs from this list. See the “Gestalt.h” file on the most recent ETO CD for a list of current gestalt 'mach' IDs.
Machines in List Are:	These radio buttons specify whether or not the computer models in the machine ID list (see above) are supported for installation.

IMPORTANT

There is only ONE machine ID list — selecting a different radio button does not give you a different list. You must decide whether to specify **either** supported machines **or** unsupported machines.

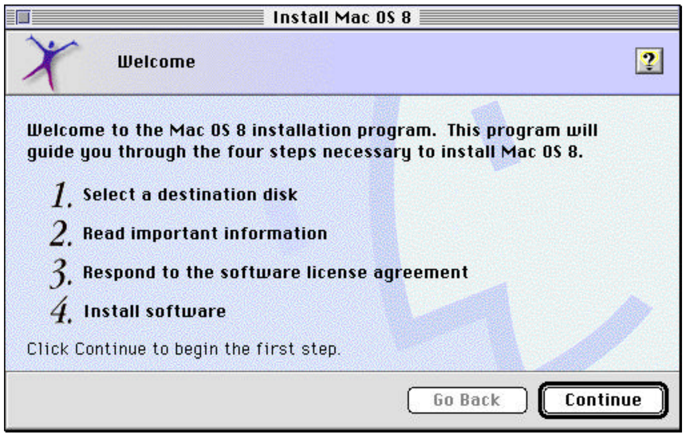
Machine ID	Enter a new machine ID number you wish to add to the ID list. If you enter an ID number that is not in the list, the “<< Add” button will be enabled.
Remove	Removes the selected ID from list.
<< Add	Adds the ID in the “Machine ID” field to the list if it doesn’t already exist.
Remove	Deletes the environment filter plug-in resources from the data file and closes the editor window.
Cancel	Closes the editor window without updating the environment filter plug-in resources.
Save	Updates the environment filter plug-in resources in the data file with the contents of the window.

Editing the Welcome Plug-in

The welcome plug-in presents a single panel for displaying a combined graphic and text message that describes the rest of the user experience. Figure 2-8 shows the Welcome plug-in panel in the Install Mac OS 8 program.

Plug-in file name: Welcome

Figure 2-8 Welcome plug-in panel



Welcome to the Mac OS 8 installation program. This program will guide you through the four steps necessary to install Mac OS 8.

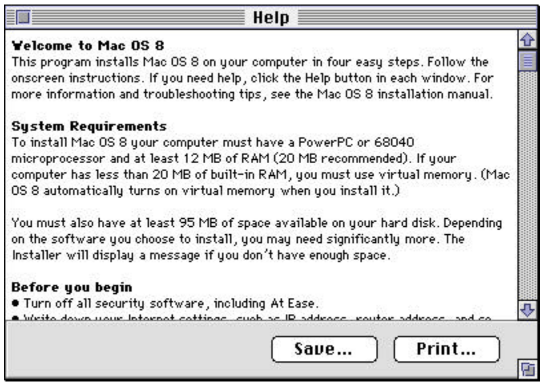
1. Select a destination disk
2. Read important information
3. Respond to the software license agreement
4. Install software

Click Continue to begin the first step.

The Install Mac OS 8 welcome plug-in uses text superimposed on top of a graphic (smiling Mac OS dude with the numbers 1 through 4).

Most plug-ins contain a help window (Figure 2-9), which is accessible via the Help icon button( ).

Figure 2-9 Help window



Welcome to Mac OS 8
This program installs Mac OS 8 on your computer in four easy steps. Follow the onscreen instructions. If you need help, click the Help button in each window. For more information and troubleshooting tips, see the Mac OS 8 installation manual.

System Requirements
To install Mac OS 8 your computer must have a PowerPC or 68040 microprocessor and at least 12 MB of RAM (20 MB recommended). If your computer has less than 20 MB of built-in RAM, you must use virtual memory. (Mac OS 8 automatically turns on virtual memory when you install it.)

You must also have at least 95 MB of space available on your hard disk. Depending on the software you choose to install, you may need significantly more. The Installer will display a message if you don't have enough space.

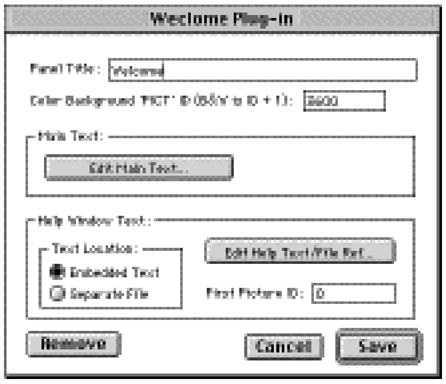
Before you begin
• Turn off all security software, including At Ease.

The content of the help window is defined by the developer or client. If the user leaves the help window open while moving to another plug-in, the content will automatically be updated with the next plug-in's help text.

Using the Welcome Plug-in Editor

The welcome editor is shown in Figure 2-10. This editor allows the developer or client to change panel text, graphic, and help text.

Figure 2-10 Welcome editor window



Weclome Plug-in

Panel Title: Welcome

Color Background 'PICT' ID (B&W is ID + 1): 8800

Main Text:

Edit Main Text...

Help Window Text:

Text Location:

(●) Embedded Text

( ) Separate File

Edit Help Text/File Ref...

First Picture ID: 0

Remove   Cancel   Save

The items in the welcome plug-in editor window are explained in detail below.

Panel Title	Enter the name that shows at the top of the panel.
Color Background 'PICT' ID	Enter the ID of the 'PICT' resource that you wish displayed in the background of the window. To support displaying a B&W picture when the monitor is displaying less than 256 colors/grays, also add a 'PICT' resource with an ID of 1 plus the ID entered in this field. The picture must be exactly 205 pixels in height by 506 pixels in width; otherwise, the picture will be scaled to fit the panel. ModifierTool does not provide a facility for adding, removing or changing 'PICT' resources inside the data file. Use your favorite resource editing program, such as ResEdit or Resorcerer, to add your pictures to the data file.
Main Text:	
Edit Main Text…	Opens the text editor window so the main text can be changed.
Help Window Text:	
Text Location:	Select Embedded Text to store the text inside the data file; otherwise, select Separate File to store the text in a text document. If you change the text location after defining the text or a file reference, this data will be deleted when saving the window, since only the text *or* a file reference is saved, not both.
Edit Help Text/File Ref…	Opens the text editor window if the text is stored in the data file; otherwise, opens the file reference window if stored in a separate text file.
First Picture ID	Enter the ID of the first 'PICT' resource embedded in the text.
Remove	Deletes the welcome plug-in resources from the data file and closes the editor window.
Cancel	Closes the editor window without updating the welcome plug-

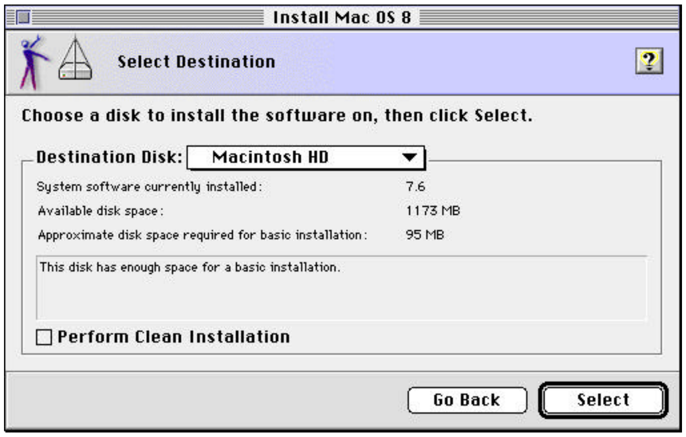
	in resources.
Save	Updates the welcome plug-in resources in the data file with the contents of the window.

Editing the Target Selection Plug-in

The target selection plug-in presents a single panel to allow the user to choose the destination disk. Figure 2-11 shows the Target Selection plug-in panel in the Install Mac OS 8 program.

Plug-in file name: TgtSelect

Figure 2-11 Target selection plug-in panel



Choose a disk to install the software on, then click Select.

Destination Disk: Macintosh HD

System software currently installed:	7.6
Available disk space:	1173 MB
Approximate disk space required for basic installation:	95 MB

This disk has enough space for a basic installation.

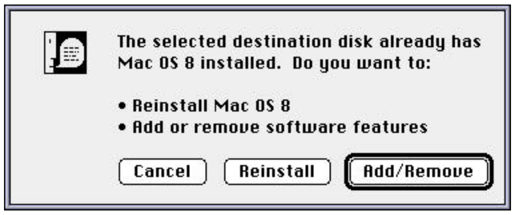
☐ Perform Clean Installation

The target selection plug-in recommends the hard drive on which the user is most likely to install. The search for a valid destination disk starts with the selected device in the Startup Device control panel (if writable), the internal hard disk, any connected SCSI hard disk, and finally any writable device. If an acceptable destination disk is not found, the disk information is hidden, the Select button is disabled, and instructions of how to proceed are displayed in the message area.

Disk information is displayed for the selected destination disk. If the blessed System Folder exists on the disk, its version number is displayed; otherwise, the text “None Installed” is displayed. In addition to the disk’s available space, the estimated disk space required for the recommend installation provided by the developer or client is displayed.

If the installation will install an entire System Folder, then the clean install option is available to the user so a new System Folder can be created, instead of upgrading the existing System Folder on the chosen destination disk. If the user chooses to install into an already upgraded System Folder, then an alert allows the user to skip directly to the custom installation panel (see Figure 2-12), bypassing the important information and software license panels. The reinstall alert will not be shown if the user has selected the clean install option.

Figure 2-12 Reinstall alert



The selected destination disk already has Mac OS 8 installed. Do you want to:

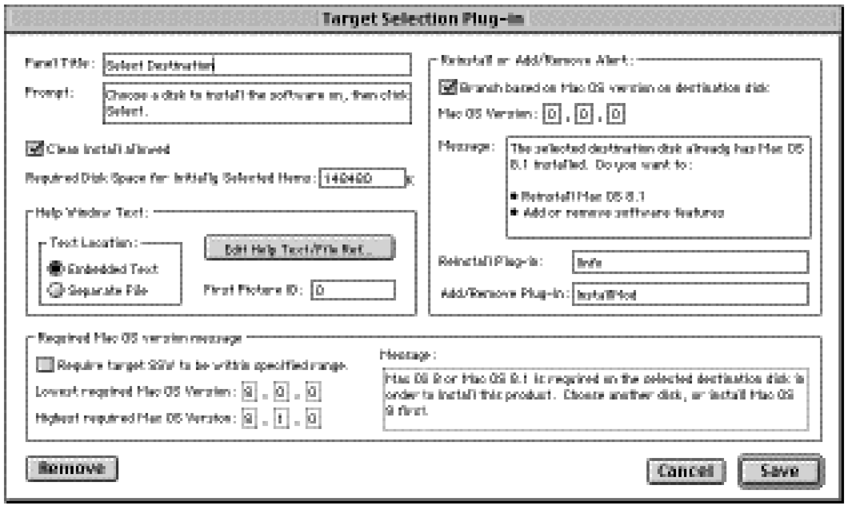
• Reinstall Mac OS 8
• Add or remove software features

Using the Target Selection Plug-in Editor

The target selection editor is shown in Figure 2-13. This editor allows the developer or client to change the following items in the target selection plug-in panel:

- panel title name
- panel prompt string
- whether a clean install is allowed.
- approximate disk space required by a recommended installation
- reinstall alert version number and text
- help window text

Figure 2-13 Target selection editor window



Target Selection Plug-in

Panel Title: Select Destination

Prompt: Choose a disk to install the software on, then click Select.

[✓] Clean install allowed

Required Disk Space for Initially Selected Items: 148480 K

Help Window Text:

Text Location: [● Embedded Text] ○ Separate File

First Picture ID: 0

Reinstall or Add/Remove Alert:

[✓] Branch based on Mac OS version on destination disk

Mac OS Version: 0 . 0 . 0

Message: The selected destination disk already has Mac OS 8.1 installed. Do you want to:

■ Reinstall Mac OS 8.1
■ Add or remove software features

Reinstall Plug-in: Info

Add/Remove Plug-in: InstallMod

Required Mac OS version message

[ ] Require target SSW to be within specified range.

Lowest required Mac OS Version: 9 . 0 . 0

Highest required Mac OS Version: 9 . 1 . 0

Message: Mac OS 9 or Mac OS 9.1 is required on the selected destination disk in order to install this product. Choose another disk, or install Mac OS 9 first.

The items in the target selection plug-in editor window are explained in detail below.

Panel Title	Enter the name that shows at the top of the panel.
Panel Prompt	Enter the text that prompts the user to perform the action of selecting an appropriate destination disk.
Clean install allowed	Select this option if the recommended installation can create a valid new System Folder.

Required Disk Space	Enter the number of kilobytes (1 kilobyte = 1024 bytes) required by the recommended installation.

NOTE

Given the differences between the disk space used on small capacity versus large capacity HFS-formatted hard drives, the value you specify must be an averaged estimation. For most products, the disk space taken by your recommended installation on a 4 GB hard disk will cover the majority of installation scenarios.

Help Window Text:

Text Location:	Select Embedded Text to store the text inside the data file; otherwise, select Separate File to store the text in a text document. If you change the text location after defining the text or a file reference, this data will be deleted when saving the window, since only the text *or* a file reference is saved, not both.
Edit Help Text/File Ref...	Opens the text editor window if the text is stored in the data file; otherwise, opens the file reference window if stored in a separate text file.
First Picture ID	Enter the ID of the first 'PICT' resource embedded in the text.

Reinstall or Add/Remove Alert:

Branch based on Mac OS version...	Select this option to present the reinstall alert so the user can jump directly to the custom installation panel when the version of the system software on the destination disk matches.
Mac OS Version	Enter the version of system software on the chosen destination disk that will trigger the reinstall alert. This should normally be the same version as the system software being installed.
Message	Enter the text to be displayed in the reinstall alert.
Reinstall Plug-in	Enter the name of the plug-in to advance to when the Reinstall button is clicked by the user.
Add/Remove Plug-in	Enter the name of the plug-in to advance to when the Add/Remove button is clicked by the user.

Required Mac OS Version Message:

Required target SSW version...	Select this option to prevent the user from continuing if the version of the system software on the chosen destination disk is not within the specified range. This option is most often used for installations that only upgrade specific versions of Mac OS.
Lowest Mac OS Version	Enter the minimal required version of system software on the chosen destination.
Highest Mac OS Version	Enter the maximum required version of system software on the chosen destination disk. This should normally be the same version as the system software being installed.

Message	Enter the text to be displayed in the message area of the target selection panel.
Remove	Deletes the target selection plug-in resources from the data file and closes the editor window.
Cancel	Closes the editor window without updating the target selection plug-in resources.
Save	Updates the target selection plug-in resources in the data file with the contents of the window.

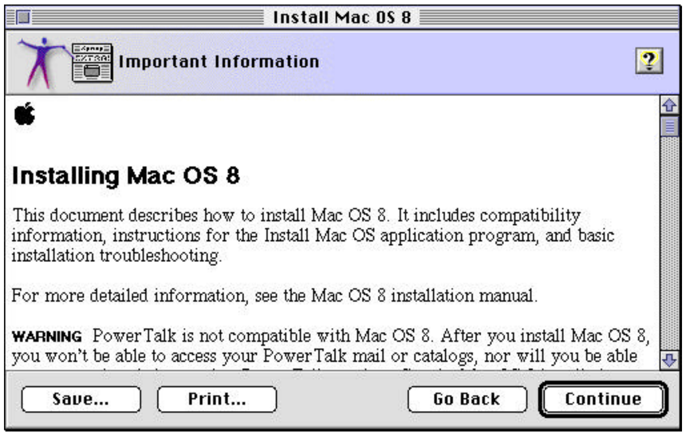
Editing the Read Me Plug-in

The read me plug-in provides a scrollable text message area for presenting important information for the user. Figure 2-14 shows the read me plug-in panel in the Install Mac OS 8 program.

Plug-in file name: IInfo

The Save and Print buttons are enabled if the user can save and print the text. When saving, a SimpleText document is created.

Figure 2-14 Read Me plug-in panel



Installing Mac OS 8

This document describes how to install Mac OS 8. It includes compatibility information, instructions for the Install Mac OS application program, and basic installation troubleshooting.

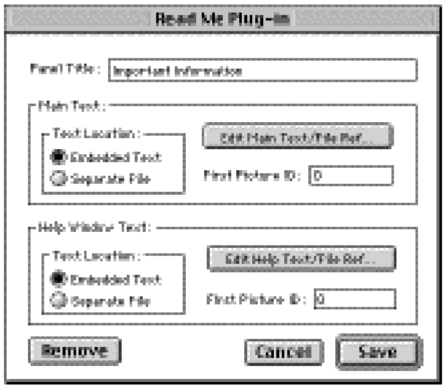
For more detailed information, see the Mac OS 8 installation manual.

WARNING PowerTalk is not compatible with Mac OS 8. After you install Mac OS 8, you won't be able to access your PowerTalk mail or catalogs, nor will you be able

Using the Read Me Plug-in Editor

The read me editor is shown in Figure 2-15. This editor allows the developer or client to change the panel title, text, and help text.

Figure 2-15 Read Me editor window



Panel Title: Important Information

Main Text: Text Location: ● Embedded Text ○ Separate File — First Picture ID: 0

Help Window Text: Text Location: ● Embedded Text ○ Separate File — First Picture ID: 0

The items in the Read Me plug-in editor window are explained in detail below.

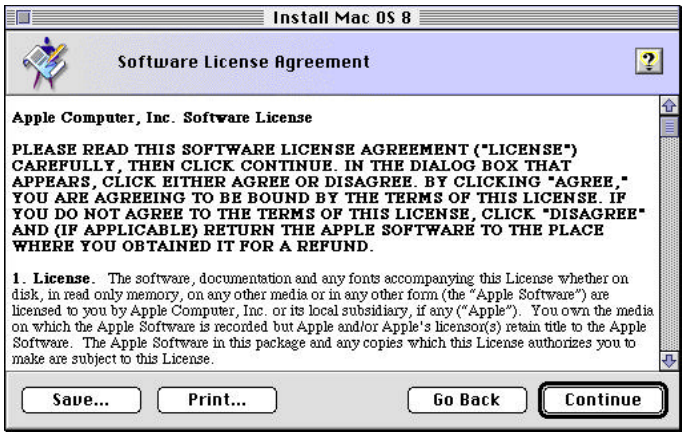
Panel Title	Enter the name that shows at the top of the panel.
Main Text:	
Text Location:	Select Embedded Text to store the text inside the data file; otherwise, select Separate File to store the text in a text document. If you change the text location after defining the text or a file reference, this data will be deleted when saving the window, since only the text *or* a file reference is saved, not both.
Edit Main Text/File Ref…	Opens the text editor window if the text is stored in the data file; otherwise, opens the file reference window if stored in a separate text file.
First Picture ID	Enter the ID of the first 'PICT' resource embedded in the text.
Help Window Text:	
Text Location:	Select Embedded Text to store the text inside the data file; otherwise, select Separate File to store the text in a text document. If you change the text location after defining the text or a file reference, this data will be deleted when saving the window, since only the text *or* a file reference is saved, not both.
Edit Help Text/File Ref…	Opens the text editor window if the text is stored in the data file; otherwise, opens the file reference window if stored in a separate text file.
First Picture ID	Enter the ID of the first 'PICT' resource embedded in the text.
Remove	Deletes the read me plug-in resources from the data file and closes the editor window.
Cancel	Closes the editor window without updating the read me plug-in resources.
Save	Updates the read me plug-in resources in the data file with the contents of the window.

Editing the Software License Plug-in

The software license plug-in presents a single panel to allow the user to read and agree to the software license before proceeding. As an option, multiple language versions of the text can be provided to address certain multi-country legal requirements. Figure 2-16 shows the software license plug-in panel in the Install Mac OS 8 program.

Plug-in file name: SWLicense

Figure 2-16 Software license plug-in panel



Apple Computer, Inc. Software License

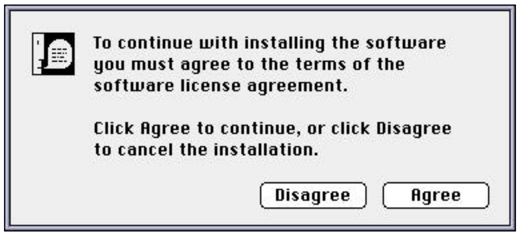
PLEASE READ THIS SOFTWARE LICENSE AGREEMENT ("LICENSE") CAREFULLY, THEN CLICK CONTINUE. IN THE DIALOG BOX THAT APPEARS, CLICK EITHER AGREE OR DISAGREE. BY CLICKING "AGREE," YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, CLICK "DISAGREE" AND (IF APPLICABLE) RETURN THE APPLE SOFTWARE TO THE PLACE WHERE YOU OBTAINED IT FOR A REFUND.

1. License. The software, documentation and any fonts accompanying this License whether on disk, in read only memory, on any other media or in any other form (the "Apple Software") are licensed to you by Apple Computer, Inc. or its local subsidiary, if any ("Apple"). You own the media on which the Apple Software is recorded but Apple and/or Apple's licensor(s) retain title to the Apple Software. The Apple Software in this package and any copies which this License authorizes you to make are subject to this License.

A pop-up menu appears in the top-right corner of the panel when multiple language versions of the license are available.

When the user clicks Continue, the Agree / Disagree alert is presented (see Figure 2-17) to force the user to explicitly denote his or her acceptance or non-acceptance to the license.

Figure 2-17 Agree/Disagree alert



To continue with installing the software you must agree to the terms of the software license agreement.

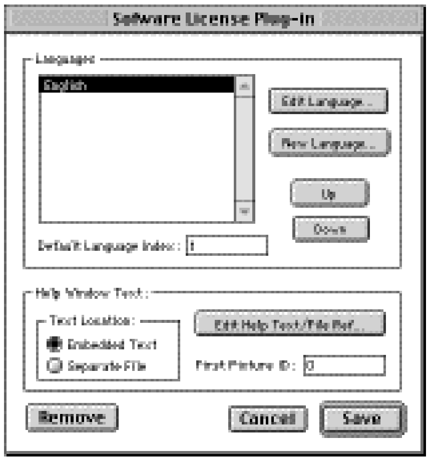
Click Agree to continue, or click Disagree to cancel the installation.

If the user clicks Agree in the Agree / Disagree alert, the user advances to the next panel. If the user clicks Disagree, then the user goes to a panel defined by the developer. In the case of the Install Mac OS 8 program, then user is taken back to the Welcome panel.

Using the Software License Plug-in Editor

The software license editor is shown in Figure 2-18. This editor allows the developer or client to change the license text and add / subtract the provided languages of the license text.

Figure 2-18 Software license editor window



Default Language Index: 1

[● Embedded Text] ○ Separate File

First Picture ID: 0

The items in the software license plug-in editor window are explained in detail below.

Languages:

Edit Language…	Shows the language editor window for the selected language, so attributes of the language can be changed. Double-clicking on a language name is a shortcut for clicking the Edit Language button.
New Language…	Shows an empty language editor window. If the user clicks OK the new language entry will added to the languages list. If a language is selected in the list when clicking New Language, the new language will be inserted before the selected language; otherwise, it will be added to the end of the list.
Up	Moves the selected entry one position closer to the beginning of the languages list.
Down	Moves the selected entry one position closer to the end of the languages list.
Default Language Index	The language entry index (starting with 1) which will be used as the default language if the running system's primary language ID does not match a language ID contained in the language list.

Help Window Text:

Text Location:	Select Embedded Text to store the text inside the data file; otherwise, select Separate File to store the text in a text document. If you change the text location after defining the text or a file reference, this data will be deleted when saving the window, since only the text *or* a file reference is saved, not both.
Edit Help Text/File Ref…	Opens the text editor window if the text is stored in the data file; otherwise, opens the file reference window if stored in a separate text file.
First Picture ID	Enter the ID of the first 'PICT' resource embedded in the text.
Remove	Deletes the software license plug-in resources from the data file and closes the editor window.

Cancel	Closes the editor window without updating the software license plug-in resources.
Save	Updates the software license plug-in resources in the data file with the contents of the window.

Multilingual Limitations

Understanding the limitations of presenting multiple languages on a single system will help you make the appropriate tradeoffs when setting up the software license plug-in. The first question to ask is what the absolute minimum legal requirements for your software license are. Normally, you'll choose one of the following:

- Only one language is required.
- Multiple single-byte character language text is required, but without multilingual controls.
- Multiple single-byte and two-byte character language text is required, but without multilingual controls.
- Multiple single-byte character language text with multilingual controls is required.
- Multiple single-byte and two-byte character language text with multilingual controls is required.

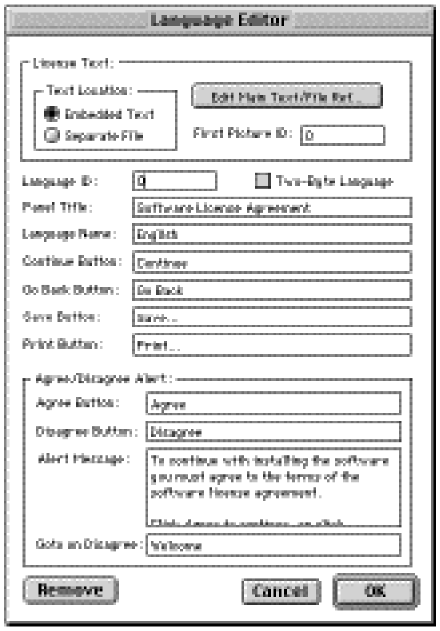
You will want to select the option with the least amount of multilingual complexity because of two fundamental limitations of the system software: the fonts installed may not support the language being displayed, and even if a language kit installed allows for correct display of the text the controls may not be displayed correctly.

Since you will not be able to depend on having fonts installed that fully support every language, even presenting multiple single-byte character languages together may be a problem. For example, the U.S. version of the Geneva font does not contain some of the accented characters that appear in the French version of that font. This limits the characters that can be used without causing boxes to appear when the character doesn't exist.

For systems that have one or more language kits installed, the software license text should be drawn correctly, but the button text will be drawn as "garbage" characters. A workaround for this problem is to leave the buttons of such languages in the main language of the Upgrader application.

As you can see, this version of the software license does not solve all the problems inherit in displaying multiple languages. The implementation of the software license plug-in has been designed to address minimal legal requirements, instead of providing a full-featured multilingual facility. This is why we suggest that you consider these limitations carefully when deciding which languages to mix.

Figure 2-19 Language editor window


Language Editor

License Text:

Text Location: [✓ Embedded Text] Separate File

Edit Main Text/File Ref...

First Picture ID: 0

Language ID: 0

Two-Byte Language

Panel Title:	Software License Agreement
Language Name:	English
Continue Button:	Continue
Go Back Button:	Go Back
Save Button:	Save...
Print Button:	Print...

Agree/Disagree Alert:

Agree Button:	Agree
Disagree Button:	Disagree
Alert Message:	To continue with installing the software you must agree to the terms of the software license agreement.
Goto on Disagree:	Welcome

The items in the language editor window are explained in detail below.

License Text:

Text Location:	Select Embedded Text to store the text inside the data file; otherwise, select Separate File to store the text in a text document. If you change the text location after defining the text or a file reference, this data will be deleted when saving the window, since only the text *or* a file reference is saved, not both.
Edit Main Text / File Ref…	Opens the text editor window if the text is stored in the data file; otherwise, opens the file reference window if stored in a separate text file.
First Picture ID	Enter the ID of the first 'PICT' resource embedded in the text.
Language ID	Enter the ID (code) of the language. This allows the software license plug-in to default to the appropriate language version of the text based on the primary language of the running system. See the “Script.h” file on the most recent ETO CD for a list of the current language codes.
Two-Byte Language	Select this option if the language defined requires a two-byte script system, such as Japanese or Chinese. This allows the software license panel to disable the language name in the pop-up menu if the system software is incapable of correctly displaying the text.
Panel Title	Enter the name that shows at the top of the panel.
Language Name	Enter the name of the language. If possible, we suggest that you use the name as written in its native language. Exceptions to this guideline are two-byte languages, which may be displayed as garbage unless written in the localized language of the program.

Continue Button	Enter the name of the Continue button.
Go Back Button	Enter the name of the Go Back button.
Save Button	Enter the name of the Save button.
Print Button	Enter the name of the Print button.
Agree / Disagree Alert:	
Agree Button	Enter the name of the Agree button.
Disagree Button	Enter the name of the Disagree button.
Message	Enter the text to be displayed in the Agree / Disagree alert.
Goto on Disagree	Enter the name of the plug-in to go back to when the user clicks Disagree.

Remove	Deletes the language entry.
Cancel	Closes the language editor window without updating the language entry.
OK	Updates the language entry with the contents of the window.

Editing the Installation Plug-in

The installation plug-in allows the user to select a set of software products to install and to begin the actual installation process. This section discusses the three main user-visible windows making up the installation plug-in: the easy installation panel, the custom installation panel, and the options dialog.

Plug-in file name: InstallMod

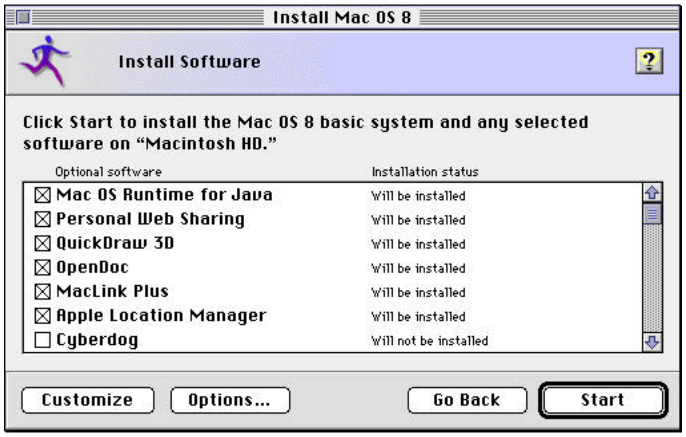
IMPORTANT

You must use version 4.0.8 or later of the Apple Installer with version 1.1.1 of the installation plug-in.

Easy Installation Panel

The easy installation panel allows the developer to provide the user with a choice of optional software products to be installed in addition to the required base set of software (see Figure 2-20). When the user clicks Start, the required software installers are run first, then any selected optional software installers are run. If the installer to be run is the Apple Installer, then an easy installation is automatically performed without user intervention.

Figure 2-20 Easy Installation panel



Click Start to install the Mac OS 8 basic system and any selected software on "Macintosh HD."

Optional software	Installation status
☒ Mac OS Runtime for Java	Will be installed
☒ Personal Web Sharing	Will be installed
☒ QuickDraw 3D	Will be installed
☒ OpenDoc	Will be installed
☒ MacLink Plus	Will be installed
☒ Apple Location Manager	Will be installed
☐ Cyberdog	Will not be installed

The software list allows the user to check the software item to be installed and provides status on the installer. The exact wording of the status description depends on whether the user is in the easy or custom installation panel, but the underlying condition is the same. The possible status descriptions are:

Will be installed	The user has selected the software item, and it will be installed when the user clicks Start.
Will not be installed	The user has not selected the software item.
Currently being installed	The software installer is currently running.
Installed successfully	The software item was successfully installed. If the user clicks Start again, this item will be skipped.
Partially installed	This is a software item that has parasites, but one or more of the parasites did not complete while this item was installing. A **parasite** software item is a special item that is always run if the previous item completes successfully. Parasites are used to fix or enhance the actions of the previous item, but are hidden from the user. If the user clicks Start again, the parasites that did not complete will be run again. (Apple Installer only)
Installation canceled	The installation was canceled by the user. If the user clicks Start again, this installer will be run again. (Apple Installer only)
Installation error	The installation was stopped because of an error the Apple Installer encountered, such as a locked file or lack of disk space. If the user clicks Start again, this installer will be run again. (Apple Installer only)

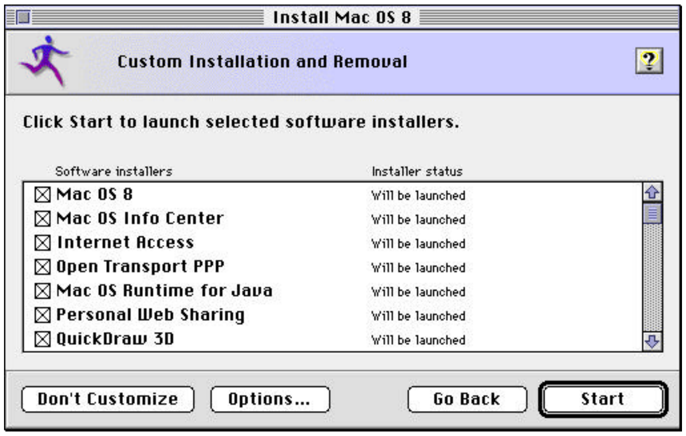
If the user goes to the previous plug-in, the status of all software items will be reset to their original default values.

The additional options available to the user depend on the developer. The Customize button takes the user to the custom installation panel, and the Options button opens the options dialog.

Custom Installation Panel

The custom installation panel shows the required base items so the user can interact directly with each Apple Installer-based software installer (see Figure 2-21). When the installation starts, the user is allowed to interact with the Installer as if they launched it from the Finder. When the installation or removal is complete, the next installer program is launched.

Figure 2-21 Custom Installation panel



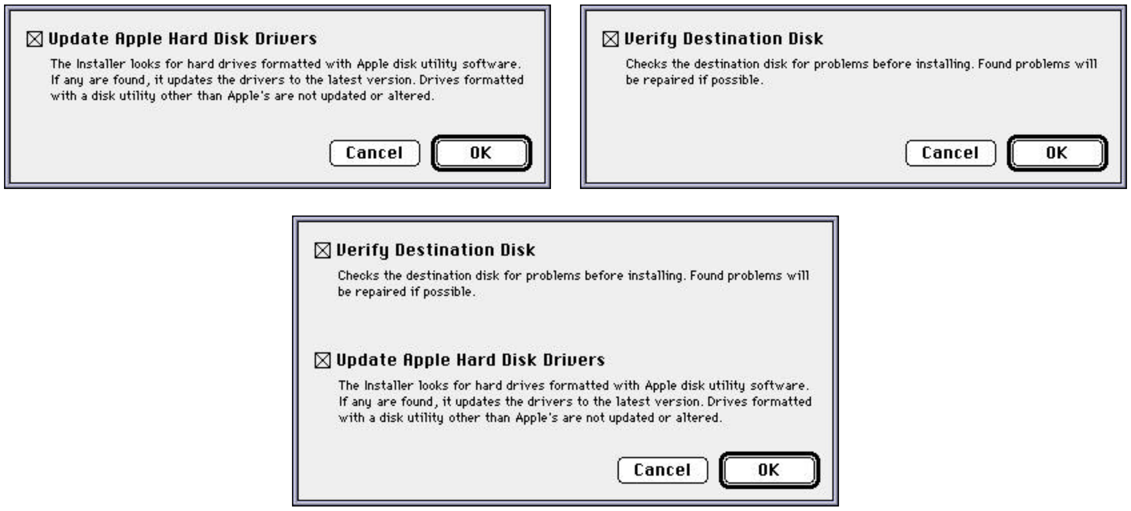
Click Start to launch selected software installers.

Software installers	Installer status
☒ Mac OS 8	Will be launched
☒ Mac OS Info Center	Will be launched
☒ Internet Access	Will be launched
☒ Open Transport PPP	Will be launched
☒ Mac OS Runtime for Java	Will be launched
☒ Personal Web Sharing	Will be launched
☒ QuickDraw 3D	Will be launched

Options Dialog

The user can change the default settings of the disk checking and disk driver updating operations from within the options dialog. The settings accessible to the user within the options dialog are defined by the developer. Figure 2-22 shows the three possible versions of the options dialog.

Figure 2-22 Three faces of the options dialog



☒ Update Apple Hard Disk Drivers

The Installer looks for hard drives formatted with Apple disk utility software. If any are found, it updates the drivers to the latest version. Drives formatted with a disk utility other than Apple's are not updated or altered.

☒ Verify Destination Disk

Checks the destination disk for problems before installing. Found problems will be repaired if possible.

☒ Verify Destination Disk

Checks the destination disk for problems before installing. Found problems will be repaired if possible.

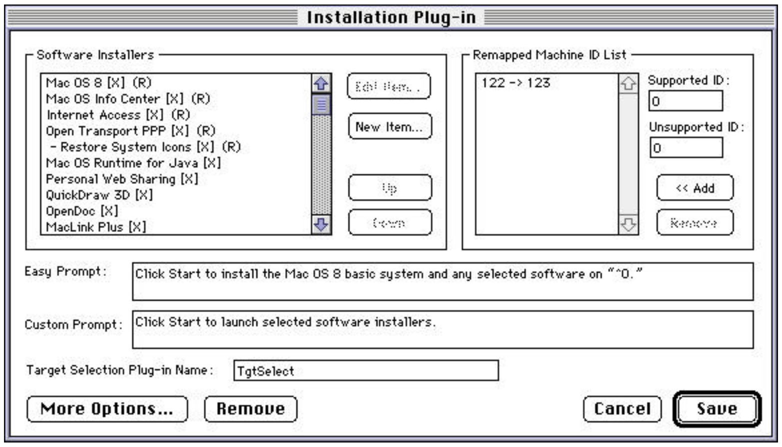
☒ Update Apple Hard Disk Drivers

The Installer looks for hard drives formatted with Apple disk utility software. If any are found, it updates the drivers to the latest version. Drives formatted with a disk utility other than Apple's are not updated or altered.

Using the Installation Plug-in Editor

The installation editor is shown in Figure 2-23. This editor allows the developer or client to change the presentation and functionality of the installation plug-in panels:

Figure 2-23 Installation editor window



Software Installers

Mac OS 8 [X] (R)
Mac OS Info Center [X] (R)
Internet Access [X] (R)
Open Transport PPP [X] (R)
 - Restore System Icons [X] (R)
Mac OS Runtime for Java [X]
Personal Web Sharing [X]
QuickDraw 3D [X]
OpenDoc [X]
MacLink Plus [X]

Remapped Machine ID List

122 -> 123

Supported ID: 0

Unsupported ID: 0

Easy Prompt: Click Start to install the Mac OS 8 basic system and any selected software on "^0."

Custom Prompt: Click Start to launch selected software installers.

Target Selection Plug-in Name: TgtSelect

The items in the installation plug-in editor window are explained in detail below.

Software Installers:

Edit Item…	Shows the software installer item window for the selected item. Double-clicking on a software item name is a shortcut for clicking the Edit Item button.
New Item…	Shows an empty software installer item window. If the user clicks OK, the new item entry will added to the software installers list. If an item is selected in the list when clicking New Item, the new item will be inserted before the selected item; otherwise, it will be added to the end of the list.
Up	Moves the selected item one entry closer to the beginning of the software installers list. If the software item has parasites, you move them up or down separately.

WARNING

Take care when reordering the software items that are marked as required. All required software items must precede the non-required software items. Any required software item listed after the first non-required software item will not be recognized by Upgrader as being required.

Down	Moves the selected item one entry closer to the end of the software items list.

Remapped Machine ID List

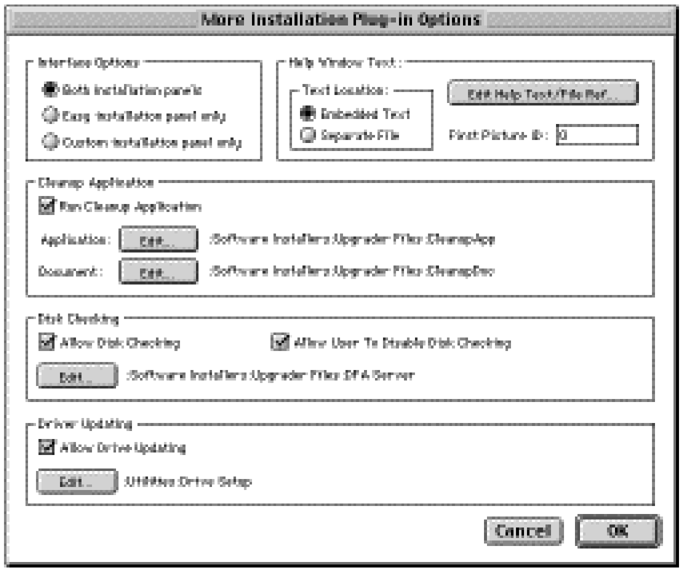
Unsupported ID	The gestalt ID of a machine that one or more of the specified Apple Installer-based Installer scripts were not designed to install on. See the "Gestalt.h" file on the most recent ETO CD

	for a list of current gestalt 'mach' IDs. The Add button is enabled if the value entered is not already in the list.
Supported ID	The gestalt ID of a machine that the unsupported machine ID should be remapped to. If a listed unsupported ID matches the ID of the machine, then its remapped ID is passed to the Apple Installer upon launch of each Installer script. The Installer script will then make decisions as if it is actually running on the older machine. See the document *Installer 4.0.7 Technical Guide* for more information about the remapping functionality.
Remove	Removes the selected ID entry from list.
<< Add	Adds the unsupported / supported ID pair to the remapped machine ID list. The list is automatically sorted by the unsupported ID value.

Easy Prompt	Enter the prompt text displayed in the easy installation panel. Since both easy and customs prompts are limited to two lines, please verify that your text fits in the actual panel.
Custom Prompt	Enter the prompt text displayed in the custom installation panel.
Target Selection plug-in name	The name of the plug-in to go back to if the destination disk unexpectedly disappears. This will usually be the target selection (“TgtSelect”) plug-in.

Remove	Deletes the installation plug-in resources from the data file and closes the editor window.
Cancel	Closes the editor window without updating the installation plug-in resources.
Save	Updates the installation plug-in resources in the data file with the contents of the window.

Figure 2-24 More installation plug-in options window



More Installation Plug-in Options

Interface Options
- [✓ Both installation panels]
- Easy installation panel only
- Custom installation panel only

Help Window Text:
- Text Location: [✓ Embedded Text] Separate File
- Edit Help Text/File Ref...
- First Picture ID: 0

Cleanup Application
- [✓] Run Cleanup Application
- Application: Edit... :Software Installers:Upgrader Files:CleanupApp
- Document: Edit... :Software Installers:Upgrader Files:CleanupDoc

Disk Checking
- [✓] Allow Disk Checking
- [✓] Allow User To Disable Disk Checking
- Edit... :Software Installers:Upgrader Files:DFA Server

Driver Updating
- [✓] Allow Drive Updating
- Edit... :Utilities:Drive Setup

Cancel OK

The items in the more installation plug-in options window are explained in detail below.

Interface Options:

Both installation panels	Select this option to show both the easy and custom installation panels.
Easy installation panel only	Select this option to show the easy installation panel only. In this case, the Customize button will be hidden.
Custom installation panel only	Select this option to show the custom installation panel only. In this case, the Don't Customize button will be hidden.

Cleanup Application:

Run Cleanup Application	Select this option to run the clean up application following the successful completion any installation.
Edit…	Click to define the file location of the clean up application.

Disk Checking:

Allow Disk Checking	Select this option to run the disk checking application before starting the actual installation.
Run Cleanup Application	Select this option to enable the user to turn off the disk checking operation from within the options dialog.
Edit…	Click to define the file location of Apple's "DFAServer" application. Currently, "DFAServer" version 1.0 is the only supported application for this option.

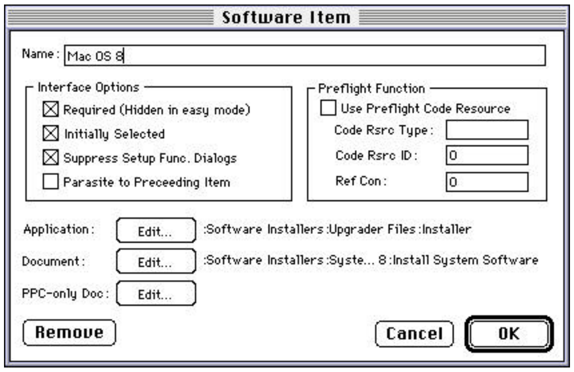
Driver Updating:

Allow Drive Updating	Select this option to run the driver updating application before starting the actual installation.
Edit…	Click to define the file location of Apple's "Drive Setup" application. Currently, "Drive Setup" version 1.3 or later is the only supported application for this option.

Help Window Text:

Text Location:	Select Embedded Text to store the text inside the data file; otherwise, select Separate File to store the text in a text document. If you change the text location after defining the text or a file reference, this data will be deleted when saving the window, since only the text *or* a file reference is saved, not both.
Edit Help Text/File Ref…	Opens the text editor window if the text is stored in the data file; otherwise, opens the file reference window if stored in a separate text file.
First Picture ID	Enter the ID of the first 'PICT' resource embedded in the text.
Cancel	Closes the more installation plug-in options window, discarding any changes made in the window.
OK	Records the changes made in the window.

Figure 2-25 Software item window



```
Software Item
Name: Mac OS 8
Interface Options
[X] Required (Hidden in easy mode)
[X] Initially Selected
[X] Suppress Setup Func. Dialogs
[ ] Parasite to Preceeding Item
Preflight Function
[ ] Use Preflight Code Resource
    Code Rsrc Type:
    Code Rsrc ID: 0
    Ref Con: 0
Application:  Edit...  :Software Installers:Upgrader Files:Installer
Document:     Edit...  :Software Installers:Syste... 8:Install System Software
PPC-only Doc: Edit...
Remove                                   Cancel   OK
```

The items in the software item window are explained in detail below.

Name	Enter the name of the item as you wish it to appear to the user in the software list.

Interface Options:

Required	Select this option to make this item part of the required base installation. The software item will be hidden in the easy installation panel and will appear as initially selected in custom installation panel.
Initially selected	Select this option to have the software item initially selected in both the easy and custom installation panels. (That is, unless it is also required — see above.)
Suppress Setup Func. Dialogs	Select this option to suppress the Installer Script's software license window (if it has one), or any window displayed from within the setup function code resource. See the document *Installer 4.0.7 Technical Guide* for more information about the setup function.
Parasite to Preceding	Select this option to signify that the software installer item is a parasite of the preceding item. Parasites will be hidden from

	the user and run whenever the software installer preceding it (another parasite or a host) successfully completes. This feature enables a developer to patch an installer. The first software item cannot be a parasite.
Preflight Function:	
Code Rsrc Type	The resource type of the preflight code resource.
Code Rsrc ID	The resource ID of the preflight code resource.
Code Rsrc RefCon	A 4-byte value passed to the preflight code resource.
Application	Click Edit to define the file location of the installer program. Normally, this is the Apple Installer, but can be any application.

NOTE

The Apple Installer and its Installer script must be located on the same source disk.

Document	Click Edit to define the file location of the installer document that support 68K- or PPC-based computers. If not file is defined for this field, but the PPC Only Document field has been define, then the installation plug-in automatically hides this software item when running on a 68K-based computer.
PPC Only Document	Click Edit to define the file location of the installer document to use when running on a PowerPC-based computer. If both document fields are defined, then the installation plug-in automatically selects the document based on the processor type.
Remove	Deletes the software installer item.
Cancel	Closes the software installer item window without updating the software installer item.
OK	Updates the software installer item with the contents of the window.

Extending the Installation Plug-in

Creating a Preflight Function

A preflight function attached to a software item can determine if the item should be hidden or checked at runtime, overriding the default attributes specified in the data file. This might be necessary if a specific installer program is designed to work on a subset of the environments the Upgrader-based program supports.

Developers can perform the following runtime decisions:

- Show or hide the software item in the easy installation panel, custom installation panel, or both panels. This allows one or more installers to be used to install a single product, each designed for a particular environment. To handle this case, the developer just needs to write a preflight function to show the appropriate one. The installation plug-in already uses this strategy for implementing its built-in support for running the correct Apple Installer script based on the processor type (68K or PPC) using the two document fields provided for each software item.

- Check or uncheck a software item based on an environmental factor. For example, the Install Mac OS 8 Upgrader-based program initially selects the Apple Location Manager item when running on PowerBooks.

Routine Definition:

```
// Result definition
enum {
   kNoError          = 0,
   kInternalError    = -1
};

typedef SInt32 SoftwareInstallerPreflightResult;

// Parameter block definition
struct SoftwareInstallerPreflightPBRec {

   // Fields set on entry
   SInt16        fDestinationVRefNum;
   SInt32        fRefCon;
   Boolean       fDoingCleanInstall;

   // Fields set by you on exit
   Boolean       fSkipOnEasy;
   Boolean       fSkipOnCustom;
   Boolean       fOverrideDefaultSelection;
   Boolean       fSelectIfOverridden;
};]

typedef struct SoftwareInstallerPreflightPBRec SoftwareInstallerPreflightPBRec,
*SoftwareInstallerPreflightPBPtr;
```

Field Descriptions:

fDestinationVRefNum	Volume RefNum of selected destination disk.
fRefCon	32-bit value passed to the code resource from data file.
fDoingCleanInstall	True if user has specified a clean install.
fSkipOnEasy	If true, forces the software installer to be hidden and skipped when running in Easy HI mode.
fSkipOnCustom	If true, forces software installer to be hidden and skipped when running in Custom HI mode.
fOverrideDefaultSelection	If true, forces initial selection state of software installer checkbox to that specified in the fSelectIfOverridden field.
fSelectIfOverridden	Selection state of software installer checkbox if fOverrideDefaultSelection field is true.

Creating a Cleanup Application

The cleanup application is run after any successful installation to perform post-installation disk modifications. It’s best to perform all cleanup operations within the software installers themselves, but sometimes you may not have control over the actions of the installers and will need a way to clean up the disk after the installation. Since the cleanup application is given no indication of which installers were run, your cleanup tasks must handle all possible installation combinations.

The cleanup application is launched with two additional parameters in the 'oapp' or 'odoc' Apple event — the selected destination disk and the process serial number of the Upgrader application. After being launched, the cleanup application should perform its tasks, then send a conclusion Apple event back to the installation plug-in before quitting.

The following parameters are included by the installation plug-in with the 'oapp' or 'odoc' Apple event upon launch of the cleanup application:

Target volume	The vRefNum of the selected destination disk. Parameter keyword: ' vtgt' Data type: typeShortInteger
Upgrader process serial number	The process serial number of the Upgrader application, so you can send the conclusion Apple event back to the Upgrader application. This parameter is included because the address you receive in the keyAddressAttr or keyOriginalAddressAttr parameters will always be the Finder. Parameter keyword: 'spsn' Data type: typeProcessSerialNumber

A conclusion Apple event that must be sent to the installation plug-in (using the Upgrader's address) by the cleanup application before the cleanup application quits:

Event class: 'pma!'; Event ID: 'revl'. No additional parameters are necessary.

Launching Other Applications

Most any application can be specified to be launched by the installation plug-in to perform the installation tasks for a software item. The application will be launched by the installation plug-in with an 'oapp' Apple event if no document is specified, or an 'odoc' Apple event if a document is specified. When the installation plug-in noticed the application has quit, it sets the status of the item to "Installed successfully" and launches the next software item, if any.

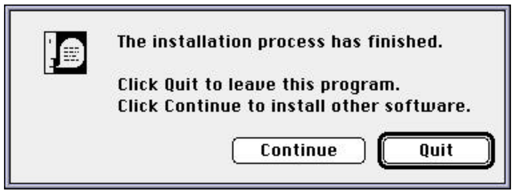
If the application being launched will force other applications to be quit or will force a restart when it completes, then it must be the last software item in the list.

Editing the Conclusion Plug-in

The conclusion plug-in displays an alert informing the user that the process is complete, but gives the user the option to go back and perform steps again, if necessary. Figures 2-26 and 2-27 show the conclusion plug-in alerts in the Install Mac OS 8 program.

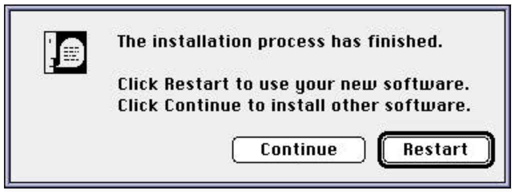
Plug-in file name: Conclusion

Figure 2-26 Conclusion plug-in alert with default quit message


The installation process has finished.

Click Quit to leave this program.
Click Continue to install other software.

Figure 2-27 Conclusion plug-in alert with default restart message

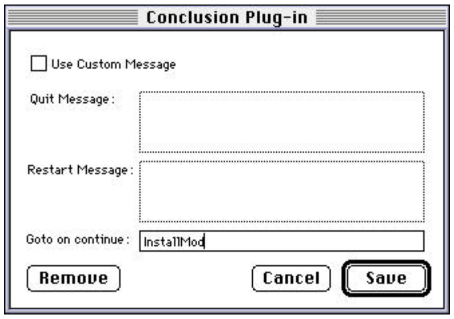

The installation process has finished.

Click Restart to use your new software.
Click Continue to install other software.

Using the Conclusion Plug-in Editor

The conclusion editor is shown in Figure 2-28. This editor allows the developer or client to change the message shown in the conclusion alert and the plug-in to go back to when the user clicks Continue.

Figure 2-28 Conclusion editor window



InstallMod

The items in the conclusion plug-in editor window are explained in detail below.

Use Custom Message	Select this option to override the default text and use the text provided in the Quit Message and Restart Message field in the alert. You may want to provide you own custom messages if the default messages not appropriate in the context of your Upgrader-based program.
Quit Message	Enter the custom message that will show in the quit alert. The default quit message is: “The installation process has finished. Click Quit to leave this program. Click Continue to install other software.”
Restart Message	Enter the custom message that will show in the restart alert. The default restart message is: “The installation process has finished. Click Restart to use your new software. Click Continue to install other software.”
Goto on continue	The name of the plug-in to go back to when the user clicks Continue in the conclusion alert. This will usually be the preceding plug-in.
Remove	Deletes the conclusion plug-in resources from the data file and closes the editor window.

Cancel	Closes the editor window without updating the conclusion plug-in resources.
Save	Updates the conclusion plug-in resources in the data file with the contents of the window.

Global Data

Target Selection Plug-in Global Data

Data types read and set:

Destination Disk	The volume refnum of the currently chosen destination disk. (Type: 'trgt', Data: 2-byte signed integer)
Clean Install Flag	A flag designating whether the clean install option has been selected by the user. (Type: 'clin', Data: 1-byte Boolean)

Data types set only:

Goto Custom Installation Flag	A flag set to true if the user clicks the Add / Remove button in the reinstall alert; otherwise, it is set to false. (Type: 'incu', Data: 1-byte Boolean)

Installation Plug-in Global Data

Data types read:

Destination disk	The volume refnum of the currently chosen destination disk. (Type: 'trgt', Data: 2-byte signed integer)
Clean install flag	If true, tells the Apple Installer for the first software item that a clean install has been requested by the user. (Type: 'clin', Data: 1-byte Boolean)
Goto custom installation flag	If true, the installation plug-in goes straight to the custom installation panel. (Type: 'incu', Data: 1-byte Boolean)

Data types set:

Restart required flag	A flag designating whether a forced restart is required or not. (Type: 'rsrq', Data: 1-byte Boolean)

Conclusion Plug-in Global Data

Data types read:

Restart required flag	A flag designating whether a forced restart is required or not. If this type is not defined, the conclusions plug-in uses the quit alert.

Resources

These resources are contained in the data file.

The Sequence Resource ('tsqc')

The sequence resource is the single most important resource used by the Upgrader. It contains the default sequence of plug-ins, specific flags for each plug-in, the IDs of the resources used to find plug-ins, the IDs of resources which are used to preload plug-in resources and other data which isn’t directly used by the plug-ins, such as the IDs of splash screen 'PICT's, etc.

```
#define    ShellFlags                                                    \
       fill bit[16]                                                      /* Reserved */

type kSequenceResourceType {
   switch {
       case format2:
          key integer = 2;                // Format version
          ShellFlags;                     // Flags
          integer BWSplashPict;           // B&W splash screen picture - 'PICT' Rsrc ID
          integer ColorSplashPict;        // 8-bit color splash screen picture - 'PICT'ID
          integer systemResListID;        // System file preload list - 'RES#' ID
          integer helpPanelResID;         // Dialog ID of help window - 'DITL' Rsrc ID
          integer defaultHelpTitleStrResID;// Help window title - 'STR ' Rsrc ID
          integer defaultHelpTextResID;   // Default help text - 'TEXT' Rsrc ID
          integer defaultHelpBasePICTResID;// First picture in help text - 'PICT' Rsrc ID
          pstring;                        // First plug-in name
          align word;
          unsigned integer = $$CountOf(pluginList);
          wide array pluginList {
             unsigned longint onlyRunOnce = 1;   // Plug-in flags
             pstring;                     // Plug-in name
             align word;
             pstring;                     // Default next plug-in name
             align word;
             unsigned longint;            // RefCon value
             unsigned integer;            // Plug-in file location - 'flrf'Rsrc ID
             unsigned integer;            // Data file preload list - 'RES#'Rsrc ID
          };
       };
};
```

Field descriptions

B&W Splash 'PICT' ID	The ID of a 'PICT' resource to be displayed when the main monitor has a color depth of less than 256 colors. Use 0 if you do not wish to provide a separate B&W picture. The dimension of your splash screen picture should be no larger than 320 pixels in height by 500 pixels in width. The Upgrader application will automatically resize the window based on the size of the picture.
Color Splash 'PICT' ID	The ID of a 'PICT' resource to be displayed when the main monitor has a color depth of 256 or more colors.
System Preload List	The ID of a 'RES#' resource containing a list of the resources that must be preloaded and marked as non-purgeable when the System file is on ejectable media, such as a floppy disk. You’ll only need to worry about this list if you ship a floppy disk set which allows the user to boot from the first floppy disk. If you do, then you may need to add System resources to this list that

	plug-ins or helper applications require during their operation. If you fail to add these additional resources to this list, extra disk swaps back to the System floppy disk will be required, which can degrade the user experience of your Upgrader-based program.
Default Help Panel Dialog ID	The ID of the 'DITL' resource of the default help window. This value should be 1050.
Default Help Window Title Ref.	The ID of a 'STR ' resource containing the title of the default help window.
Default Help Window Text Ref.	The ID of 'TEXT' (and optional 'styl') resource containing the default help text for plug-ins which do not provide their own text.
Default Help Window First Pict.	The ID of a 'PICT' resource of the first resource to be displayed in the default help text panel.
First Plug-in Name	The name of the plug-in that is to be called first. The Upgrader will use this to begin building the default plug-in sequence list.
onlyRunOnce Flag	Use the onlyRunOnce flag if the user should be prevented from going back to this plug-in. The Go Back button in the Upgrader-based program will automatically be dimmed if the preceding plug-ins are set to run once. The Environmental Filter plug-in included with the Install Mac OS 8 program uses this feature, since the user’s machine only needs to be checked once during the launch of the program.
Plug-in Name	The name of the plug-in to be added to the list.
Next Plug-in Name	The name of the default next plug-in that the Upgrader is to look for once this plug-in is finished.
RefCon Value	A 4-byte value passed to the plug-in upon initialization. Most plug-ins will want to use the low-word of this value to store the resource ID of its preference resource. This allows for a single plug-in file to be used for multiple plug-ins, each with a different resource ID.
Plug-in File Ref.	The ID of an 'flrf' resource which defines the location of the plug-in. Integer values 1 and 0 are reserved for use by the Upgrader.
System Preload List ID	The ID of a 'RES#' resource, which contains the types and IDs of all the specific plug-in resources contained within the data file which are to be preloaded. You’ll only need to worry about this list if you ship a floppy disk set which allows the user to boot from the first floppy disk. If you do, then you may need to add System resources to this list that plug-ins or helper applications require during their operation. If you fail to add these additional resources to this list, extra disk swaps back to the System floppy disk will be required, which can degrade the user experience of your Upgrader-based program.

The File Reference Resource ('flrf')

The file reference resource is the standard method of defining the location of a file on your source disks. Plug-in writers will need to create one of these for their plug-in and add it to the data file. The ID of the resource will then need to be added to that plug-in's information in the Sequence resource so that the Upgrader can find the 'flrf' resource and use it to locate the plug-in.

Plug-ins can also use the file reference resource to specify applications, documents and other files that the plug-in uses.

```
#define   kUnknownMedia     0      // Net install setup on CD-ROM. Use relative path.
#define   kFloppyDiskMedia  1      // Multiple floppy disk set. Use full path.
#define   kCDROMDiskMedia   2      // Multiple CD-ROM disk set. Use full path.

#define   FileRefFlags                                                 \
       fill bit[16]                                                    /* Reserved */

type 'flrf' {
switch {
       case format0:
              key integer = 0;                  // Format version
              FileRefFlags;                     // Flags
              literal longint;                  // File Type
              literal longint;                  // File Creator
              longint;                          // File Creation Date (optional)
              pstring;                          // File Path
              align word;
              pstring;                          // File Name
              align word;
       };
};
```

Field descriptions

Media Type	This value tells the Upgrader application how to find the specified file. If media type is kUnknownMedia, then the Upgrader-based program expects a relative path from the folder that contains the Upgrader application. If media type is kFloppyDiskMedia or kCDROMDiskMedia then the path is assumed to be a full path beginning with the name of the floppy or CD-ROM disk. To support creating a net install from a set of CD-ROM or floppy disks, the Upgrader-based program will first look in the folder containing the data file to find a folder with the same name as the CD-ROM or floppy disk before asking the user to insert the disk.
File Type	The four character “type” of the file. Use 0 to cause Upgrader to ignore the file's type when locating the file.
File Creator	The four character “creator” of the file. Use 0 to cause Upgrader to ignore the file's creator when locating the file.
File Creation Date	The creation date of the file in seconds. Use 0 to cause Upgrader to ignore the file's creation date when locating the file.
File Path	This should be a relative path from the folder containing the data file or a full path beginning with the name of the floppy or CD-ROM disk. Use of a full or relative path depends on the media type. See the “Media Type” field description for more information.

	If the path is relative, then it should start with a colon. A full path should start with the name of the floppy or CD-ROM disk, instead of a colon.
File Name	Name of the file. Limited to 31 characters.

The Resource List Resource ('RES#')

The resource list resource contains a list of resource type and ID pairs for holding the list of resources to preload. This resource is referenced from the sequence resource to hold the resources to preload from the System file and data file

Plug-in writers can also use a 'RES#' resource located in the plug-in file to preload resources stored in the plug-in file by passing the ID to the `PSCollect` routine.

```
type 'RES#' {
       integer = $$CountOf(ResArray);
       array ResArray {
              literal longint;            // Resource type to preload
              integer;                    // Resource ID to preload
              };
};
```

The Data File Format Resource ('dfmt')

The data file format resource defines the version of the client data file, so Upgrader can determine if it can read the data file or not. The user will receive a message if Upgrader cannot open data file because the format resource is incompatible or missing.

```
type 'cfmt' {
   byte DataFileMajorRevisionNumber;             // The major version number
   byte DataFileMinorRevisionNumber;             // The minor version number
};
```

Upgrader application versions and the data file versions they support:

Upgrader Versions	**Data File Versions Supported**
1.1	0.3
1.1.1	0.4

The Environmental Filter Plug-in Preference Resource ('efpr')

```
#define    EFPPrefFlags                                        \
       boolean    kMachIDListNotSupported, kMachIDListSupported; \
       fill bit[15]                       /* Reserved */

type 'efpr' {
   switch {
       case format1:
          key integer = 1;                /* Format version */
          EFPPrefFlags;                   /* Flags */
          integer;                        /* Minimum released SSW version */
          integer = $$CountOf(MachIDArray);
          array MachIDArray {
              integer;                    /* Machine Gestalt ID */
```

```
            };
        };
    };
```

kMachIDListNotSupported / kMachIDListSupported flag	Use the kMachIDListNotSupported flag if the machine IDs listed are not supported by this Upgrader-based program, and therefore should alert the user. Use the kMachIDListSupported flag if the machine IDs listed are the only computer models that are supported by this Upgrader-based program.
Minimum Released System…	The minimum Mac OS system version on which your Upgrader-based program should run. The value is in BCD format. For example, version 7.1.2 would be specified in hex as 0x0712. Since the Upgrader application requires at least system 7.0, the value must be 0x0700 or higher.
Gestalt Machine ID	A Gestalt machine ID. See the “Gestalt.h” file on the most recent ETO CD for a list of current gestalt 'mach' IDs.

Welcome Plug-in Preference Resource ('wppr')

```
#define    WPPrefFlags                                                      \
       boolean    kReserved1, kReserved2;  /* Ignored. */\
       boolean    kHelpTextInRsrc, kHelpTextInFile;      /* Location of help text. */ \
       fill bit[14]                                      /* Reserved */

type 'wppr' {
    switch {
       case format2:
          key integer = 2;  /* Format version */
          WPPrefFlags;      /* Flags */
          integer;          /* Main Text - 'flrf' or 'TEXT' Rsrc ID */
          integer;          /* Help text - 'flrf' or 'TEXT' Rsrc ID */
          integer;          /* First help text picture - 'PICT' Rsrc ID */
          integer;          /* Background color picture (B&W: ID + 1) - 'PICT' Rsrc ID */
          integer;          /* Plug-in string list Rsrc ID ('STR#') - 'STR#' Rsrc ID */
    };
};
```

kHelpTextInRsrc, kHelpTextInFile flag	Use kHelpTextInRsrc to specify that the Help Text Reference ID field points to a 'TEXT' resource. Use kHelpTextInFile to specify that the Help Text Reference ID field points to a 'flrf' resource.
Main Text Reference ID	The ID of a 'TEXT' resource if the text is stored in the data file; otherwise, the ID of a 'flrf' resource if stored in a separate text file.
Help Text Reference ID	The ID of a 'TEXT' resource if the text is stored in the data file; otherwise, the ID of a 'flrf' resource if stored in a separate text file.
Help Text First Picture ID	The ID of a 'PICT' resource containing the first picture to be embedded in the help text.
Color Background 'PICT' ID	The ID of a 'PICT' resource that you wish to be displayed in the background of the window. To support displaying a B&W

	picture when the monitor is displaying less than 256 colors/grays, also add a 'PICT' resource with an ID of 1 plus the ID entered in this field. The picture must be exactly 205 pixels in height by 506 pixels in width; otherwise, the picture will be scaled to fit the panel.
Plug-in string list Rsrc ID	The ID of a 'STR#' resource containing text strings required by the plug-in. 'STR#' resource string index definitions: 1. Panel title name 2. Help window title

Target Selection Plug-in Preference Resource ('tspr')

```
#define   TSPPrefFlags                                                    \
       boolean   kHelpTextInRsrc, kHelpTextInFile;      /* Location of help text. */ \
       boolean   kDontCheckSystemVersion, kCheckSystemVersion;             \
       boolean   kDontAllowCleanInstall, kAllowCleanInstall;               \
       boolean   kDontRequireTargetSysVersion, kRequireTargetSysVersion;   \
       fill bit[12]                                     /* Reserved */

type 'tspr' {
   switch {
       case format1:
       case format2:
           key integer = 2;         /* Format version */
           longint;                 /* Required disk space for recommended installation */
           integer;                 /* System software version being installed, if any */
           integer;                 /* Minimum Required System software version */
           integer;                 /* Maximum Required System software version */
           integer;                 /* Plug-in reference ID if reinstall - 'STR ' Rsrc ID */
           integer;                 /* Plug-in reference ID if add/remove - 'STR ' Rsrc ID */
           integer;                 /* Help text - 'flrf' or 'TEXT' Rsrc ID */
           integer;                 /* First help text picture - 'PICT' Rsrc ID */
           integer;                 /* Plug-in string list - 'STR#' Rsrc ID */
   };
};
```

kMainTextInRsrc/kMainTextInFile flag

Use kMainTextInRsrc to specify that the Main Text Reference ID field points to a 'TEXT' resource. Use kMainTextInFile to specify that the Main Text Reference ID field points to a 'flrf' resource.

kDontCheckSystemVersion/kCheckSystemVersion flag

Use kDontCheckSystemVersion to prevent the display of the reinstall alert. Use kCheckSystemVersion to display the reinstall alert if the version of the system software on the chosen destination disk matches the version in the “System software version installed” field.

kDontAllowCleanInstall/kAllowCleanInstall flag

Use kDontAllowCleanInstall to prevent the user from performing a clean installation. This would be the case if your recommended installation does not install an entire System Folder. Use kAllowCleanInstall to allow the user to perform a clean installation.

kDontRequireTargetSysVersion / kRequireTargetSysVersion flag	Use kDontRequireTargetSysVersion to ignore the version of the system software on the chosen destination disk. Use kRequireTargetSysVersion to prevent the user from continuing if the version of the system software on the chosen destination disk is not within the range specified in the “Minimum Required SSW version” and “Maximum Required SSW version” fields.
Required disk space	The number of kilobytes required by the recommended installation.

NOTE

Given the differences between the disk space used on small capacity versus large capacity HFS-formatted hard drives, the value you specify must be an averaged estimation. For most products, the disk space taken by your recommended installation on a 4 GB hard disk will cover the majority of installation scenarios.

System software version installed	The version of system software on the chosen destination disk that will trigger the reinstall alert. This should normally be the same version as system software being installed. The value is in BCD format. For example, version 8.0.1 would specified in hex as 0x0801.
Minimum Required SSW version	When using the kRequireTargetSysVersion flag, the minimum version of system software allowed on the chosen destination disk. A version lower than this value will prevent the user from continuting. The value is in BCD format. For example, version 8.0.1 would specified in hex as 0x0801.
Maximum Required SSW version	When using the kRequireTargetSysVersion flag, the maximum version of system software allowed on the chosen destination disk. A version higher than this value will prevent the user from continuting. Most of the time, the value will be the same version of system software being installed. Enter a value of 0 (zero) to not constrain the maximum version. The value is in BCD format. For example, version 8.0.1 would specified in hex as 0x0801.
Plug-in reference ID if reinstall	The ID of a 'STR ' resource containing the name of the plug-in to advance to when the user clicks the Reinstall button in the reinstall alert.
Plug-in reference ID if add / remove	The ID of a 'STR ' resource containing the name of the plug-in to advance to when the user clicks the Add / Remove button in the reinstall alert. Certain global data values are set to notify the plug-in that the user chosen this option.
Help Text Reference ID	The ID of a 'TEXT' resource if the text is stored in the data file; otherwise, the ID of 'flrf' resource if stored in a separate text file.
Help Text First Picture ID	The ID of a 'PICT' resource containing the first picture to be embedded in the help text.

Plug-in string list Rsrc ID	The ID of a 'STR#' resource containing text strings required by the plug-in. 'STR#' resource string index definitions: 1. Panel title name 2. Panel prompt text 3. Destination disk pop-up menu title 4. Help window title 5. Reinstall alert message text

Read Me Plug-in Preference Resource ('rmpr')

```
#define    RMPPrefFlags                                                       \
       boolean    kMainTextInRsrc, kMainTextInFile;      /* Help text location */     \
       boolean    kHelpTextInRsrc, kHelpTextInFile;      /* Main text location */     \
       fill bit[14]                                      /* Reserved */

type 'rmpr' {
   switch {
      case format1:
         key integer = 1;                  /* Format version */
         RMPPrefFlags;                     /* Flags */
         integer;                          /* Main Text - 'flrf' or 'TEXT' Rsrc ID */
         integer;                          /* First main text picture - 'PICT' Rsrc ID */
         integer;                          /* Help text - 'flrf' or 'TEXT' Rsrc ID */
         integer;                          /* First help text picture - 'PICT' Rsrc ID */
         integer;                          /* Plug-in string list - 'STR#' Rsrc ID */
   };
};
```

kMainTextInRsrc / kMainTextInFile flag	Use kMainTextInRsrc to specify that the Main Text Reference ID field points to a 'TEXT' resource. Use kMainTextInFile to specify that the Main Text Reference ID field points to a 'flrf' resource.
kHelpTextInRsrc, kHelpTextInFile flag	Use kHelpTextInRsrc to specify that the Help Text Reference ID field points to a 'TEXT' resource. Use kHelpTextInFile to specify that the Help Text Reference ID field points to a 'flrf' resource.
Main Text Reference ID	The ID of a 'TEXT' resource if the text is stored in the data file; otherwise, ID of 'flrf' resource if stored in a separate text file.
Main Text First Picture ID	The ID of a 'PICT' resource containing the first picture to be embedded in the main text.
Help Text Reference ID	The ID of a 'TEXT' resource if the text is stored in the data file; otherwise, the ID of 'flrf' resource if stored in a separate text file.
Help Text First Picture ID	The ID of a 'PICT' resource containing the first picture to be embedded in the help text.
Plug-in string list Rsrc ID	The ID of a 'STR#' resource containing text strings required by the plug-in.

'STR#' resource string index definitions:

1. Panel title name
2. Help window title

Software License Plug-in Preference Resource ('swpr')

```
#define    LVPPrefFlags                                                      \
       boolean    kHelpTextInRsrc, kHelpTextInFile;      /* Help text location */    \
       fill bit[15]                                      /* Reserved */


#define    LangEntryFlags                                                    \
       boolean    kOneByte, kTwoByte;             /* Language character length */    \
       boolean    kTextInRsrc, kTextInFile;       /* License text location */        \
       fill bit[14]                               /* Reserved */

type 'swpr' {

   switch {
      case format4:
         key integer = 4;         /* Format version */
         LVPPrefFlags;            /* Flags */
         integer;                 /* Help text ref - 'flrf' or 'TEXT' Rsrc ID */
         integer;                 /* First help text picture - 'PICT' Rsrc ID */
         integer;                 /* Plug-in string list ref - 'STR#' Rsrc ID */
         integer;                 /* Disagree plug-in name ref - 'STR ' Rsrc ID */
         integer;                 /* Panel title string list ref - 'STR#' Rsrc ID */
         integer;                 /* Language name string list ref - 'STR#' Rsrc ID */
         integer;                 /* Continue button string list ref - 'STR#' Rsrc ID */
         integer;                 /* Save button string list ref - 'STR#' Rsrc ID */
         integer;                 /* Print button string list ref - 'STR#' Rsrc ID */
         integer;                 /* Go Back button string list ref - 'STR#' Rsrc ID */
         integer;                 /* Agree button string list ref - 'STR#' Rsrc ID */
         integer;                 /* Disagree button string list ref - 'STR#' Rsrc ID */
         integer;                 /* Agree/Disagree dialog text ref - 'STR#' Rsrc ID */
         integer;                 /* Default language index */
         integer = $$CountOf (LanguageArray);
         array LanguageArray
            {
            LangEntryFlags;       /* Language flags */
            integer;              /* Language ID (from Script.h) */
            integer;              /* License text ref - 'flrf' or 'TEXT' Rsrc ID */
            integer;              /* First license text picture - 'PICT' Rsrc ID */
            };
   };
};
```

kHelpTextInRsrc, kHelpTextInFile flag	
	Use kHelpTextInRsrc to specify that the Help Text Reference ID field points to a 'TEXT' resource. Use kHelpTextInFile to specify that the Help Text Reference ID field points to a 'flrf' resource.
Help Text Reference ID	The ID of a 'TEXT' resource if the text is stored in the data file; otherwise, the ID of 'flrf' resource if stored in a separate text file.
Help Text First Picture ID	The ID of a 'PICT' resource containing the first picture to be embedded in the help text.

Plug-in string list ref	The ID of a 'STR#' resource containing text strings required by the plug-in. 'STR#' resource string index definitions: 1. Help window title
Disagree plug-in name ref	The ID of a 'STR ' resource containing the name of the plug-in to go back to when the user clicks Disagree in the Agree / Disagree alert.
Panel title string list ref	The ID of a 'STR#' resource containing the panel title for each language entry in the order listed.
Language name list ref	The ID of a 'STR#' resource containing the language names appearing in the pop-up menu for each language entry in the order listed.
Continue button name list ref	The ID of a 'STR#' resource containing the Continue button names for each language entry in the order listed.
Save button name list ref	The ID of a 'STR#' resource containing the Save button names for each language entry in the order listed.
Print button name list ref	The ID of a 'STR#' resource containing the Print button names for each language entry in the order listed.
Go Back button name list ref	The ID of a 'STR#' resource containing the Go Back button names for each language entry in the order listed.
Agree button name list ref	The ID of a 'STR#' resource containing the Agree button names for each language entry in the order listed.
Agree / Disagree dialog text list ref	The ID of a 'STR#' resource containing the text to be displayed in the Agree / Disagree alert for each language entry in the order listed.
Disagree button name string list ref	The ID of a 'STR#' resource containing the Disagree button names for each language entry in the order listed.
Default language index	The language entry index (starting with 1) which will be used as the default language if the running system's primary language ID does not match a language ID contained in the language list.

Language Entry:

kOneByte / kTwoByte flag	Use kOneByte if the language only uses one byte to define its characters. Use kTwoByte if the language only uses two bytes to define its characters, such as Japanese or Chinese.
kTextInRsrc, kTextInFile flag	Use kTextInRsrc to specify that the License Text Reference ID field points to a 'TEXT' resource. Use kTextInFile to specify that the License Text Reference ID field points to a 'flrf' resource.
Language ID	The ID (code) of the language. This allows the software license plug-in to default to the appropriate language version of the text based on the primary language of the system software. See the "Script.h" file on the most recent ETO CD for a list of the current language codes.

License Text Reference ID	The ID of a 'TEXT' resource if the text is stored in the data file; otherwise, ID of 'flrf' resource if stored in a separate text file.
License Text First Picture ID	The ID of a 'PICT' resource containing the first picture to be embedded in the main text.

Installation Plug-in Preference Resource ('ippr')

```
#define    IPPrefFlags                                                          \
       boolean    kDontRunCleanupApp, kRunCleanupApp;                           \
       boolean    kDontAllowEasyUpgradeMode, kAllowEasyUpgradeMode;             \
       boolean    kDontAllowCustomUpgradeMode, kAllowCustomUpgradeMode;         \
       boolean    kHelpTextInRsrc, kHelpTextInFile;                             \
       boolean    kDontUpdateDrivers, kUpdateDrivers;                           \
       boolean    kDontCheckDisk, kCheckDisk;                                   \
       boolean    kDontAllowUserToTurnOffCheckDisk, kAllowUserToTurnOffCheckDisk;     \
       fill bit[9]                                       /* Reserved */

#define    SubTaskFlags                                                         \
       boolean    kOptionalSubTask, kRequiredSubTask;                           \
       boolean    kDontSelectInitially, kSelectInitially;                       \
       boolean    kReserved1, kReserved2;                                       \
       boolean    kReserved3, kReserved4;                                       \
       boolean    kDontSuppressSetupFuncDlgSubTask, kSuppressSetupFuncDlgSubTask;     \
       boolean    kStandAloneInstallerScript, kForceInstallWithPriorInstallerScript;  \
       boolean    kAlertUserSrcDoesNotExist, kSkipIfSrcDoesNotExist;            \
       fill bit[9]                                       /* Reserved */

type 'ippr' {
   switch {
       case format1:
          key integer = 1;         /* Format version */
          IPPrefFlags;             /* Flags */
          integer;                 /* General Strings STR# Rsrc ID */
          integer;                 /* Target selection plug-in name - 'STR ' Rsrc ID */
          integer;                 /* Software item names list - STR# Rsrc ID */
          integer;                 /* DFAServer application ref - 'flrf' Rsrc ID*/
          integer;                 /* Drive Setup application ref - 'flrf' Rsrc ID */
          integer;                 /* Cleanup application ref - 'flrf' Rsrc ID */
          integer;                 /* Cleanup application document ref - 'flrf' Rsrc ID */
          integer;                 /* Help text - 'flrf' or 'TEXT' Rsrc ID */
          integer;                 /* First help text picture - 'PICT' Rsrc ID */
          integer = $$CountOf ( RemapIDPairs );
          wide array RemapIDPairs{
             integer;              /* Unsupported machine ID */
             integer;              /* Supported machine ID */
          };
          integer = $$CountOf ( SubTask );
          wide array SubTask{
             SubTaskFlags;         /* Software Item Flags */
             integer;              /* Installer application ref - 'flrf' Rsrc ID */
             integer;              /* Installer document ref - 'flrf' Rsrc ID */
             integer;              /* Installer document ref (PPC only) - 'flrf' Rsrc ID */
             literal longint;      /* Preflight code resource type */
             integer;              /* Preflight code resource ID - 'pffn' Rsrc ID */
             longint;              /* Preflight code resource refcon */
          };
   };
};
```

kDontRunCleanupApp / kRunCleanupApp flag

Use kDontRunCleanupApp to not run the clean up application. Use kRunCleanupApp to run the application specified in the Cleanup application ref field after any installations are complete.

kDontAllowEasyUpgradeMode / kAllowEasyUpgradeMode flag
Use kDontAllowEasyUpgradeMode to hide the easy installation panel. Use kAllowEasyUpgradeMode to allow the user to see the easy installation panel.

kDontAllowCustomUpgradeMode / kAllowCustomUpgradeMode flag
Use kDontAllowCustomUpgradeMode to hide the custom installation panel. Use kAllowCustomUpgradeMode to allow the user to see the custom installation panel.

kHelpTextInRsrc, kHelpTextInFile flag
Use kHelpTextInRsrc to specify that the Help Text Reference ID field points to a 'TEXT' resource. Use kHelpTextInFile to specify that the Help Text Reference ID field points to a 'flrf' resource.

kDontUpdateDrivers / kUpdateDrivers flag
Use kDontUpdateDrivers to not run the driver updating program. Use kUpdateDrivers to run the application specified in the Drive Setup application ref field after the user clicks Start in the installation panel.

kDontCheckDisk / kCheckDisk flag
Use kDontCheckDisk to not run the disk checking program. Use kCheckDisk to run the application specified in the DFAServer application ref field after the user clicks Start in the installation panel.

kDontAllowUserToTurnOffCheckDisk / kAllowUserToTurnOffCheckDisk flag
Use kDontAllowUserToTurnOffCheckDisk to hide the disk checking checkbox in the options dialog, thereby forcing the user to check the disk before each installation. Use kAllowUserToTurnOffCheckDisk to show the disk checking checkbox in the options dialog.

Plug-in string list Rsrc ID	The ID of a 'STR#' resource containing text strings required by the plug-in. 'STR#' resource string index definitions: 1. Easy installation panel prompt 2. Custom installation panel prompt
Target selection plug-in ref ID	The ID of a 'STR ' resource containing the name of the Target Selection plug-in to go back to if the destination disk unexpectedly disappears.
Software item names list ref ID	The ID of a 'STR#' resource containing names of the software items as they are displayed to the user. Each name occupies a separate index, corresponding to the order in which the software items are list.
DFAServer application ref	The ID of a 'flrf' resource containing the file location of Apple's “DFAServer” application.
Drive Setup application ref	The ID of a 'flrf' resource containing the file location of Apple's “Drive Setup” application.

Cleanup application ref	The ID of a 'flrf' resource containing the file location of an application to be run after any successful installation to perform clean up-type operations. It's normally best to perform all clean up operations within the software installers themselves, instead of referencing a separate application here.
Cleanup application document ref	The ID of a 'flrf' resource containing the file location of a document to be opened by the cleanup application. Use 0 if no document is necessary.
Help Text Reference ID	The ID of a 'TEXT' resource if the text is stored in the data file; otherwise, the ID of 'flrf' resource if stored in a separate text file.
Help Text First Picture ID	The ID of a 'PICT' resource containing the first picture to be embedded in the help text.

Remap Entry:

Unsupported machine ID	The gestalt ID of a machine that one or more of the specified Apple Installer-based Installer scripts were not designed to install on. (See the “Gestalt.h” file on the most recent ETO CD for a list of current gestalt 'mach' IDs.)
Supported machine ID	The gestalt ID of a machine that the unsupported machine ID should be remapped to. If a listed unsupported ID matches the ID of the machine, then its remapped ID is passed to the Apple Installer upon launch of each Installer script. The Installer script will then make decisions as if it is actually running on the older machine. See the document *Installer 4.0.7 Technical Guide* for more information about the remapping functionality.

Software Installer Item:

kOptionalSubTask / kRequiredSubTask flag

Use kOptionalSubTask to allow the user to optional selected this item in the easy installation panel. Use kRequiredSubTask to make this item part of the required base installation. The software item will be hidden in the easy installation panel and will appear as initially selected in custom installation panel.

kDontSelectInitially / kSelectInitially flag

Use kDontSelectInitially to have the software item default to unchecked. Use kSelectInitially to have the software item initially selected in both the easy and custom installation panels. (That is, unless it is also required — see above.)

kDontSuppressSetupFuncDlgSubTask / kSuppressSetupFuncDlgSubTask flag

Use kDontSuppressSetupFuncDlgSubTask to have the Installer script show its own software license window, if it has one. Use kDontSuppressSetupFuncDlgSubTask to suppress the Installer Script's software license window, or any window displayed from within the setup function code resource. See the document *Installer 4.0.7 Technical Guide* for more information about the setup function.

kStandAloneInstallerScript / kForceInstallWithPriorInstallerScript flag

Use kStandAloneInstallerScript for standard, non parasite software installers. Use kStandAloneInstallerScript to signify that the software installer item is a parasite of the preceding item. Parasites will be hidden from the user and run whenever the software installer preceding it (another parasite or a host)

	successfully completes. This feature enables a developer to patch an installer. The first software item cannot be a parasite.

kAlertUserSrcDoesNotExist / kSkipIfSrcDoesNotExist flag

Use kAlertUserSrcDoesNotExist to require that the specified Installer script be present on the source. If it is not found, then an alert is presented stopping the installation. Use kSkipIfSrcDoesNotExist to make the Installer script optional. If the Installer scrip is not found, the software item is automatically skipped. This might be handy if you want to create several versions of your source disk, but be able to use the same data file on each.

Installer application ref	The ID of a 'flrf' resource containing the file location of the installer program. Normally, this is the Apple Installer, but can be any application.

NOTE

The Apple Installer and its Installer script must be located on the same source disk.

Installer doc ref (68K or fat)	The ID of a 'flrf' resource containing the file location of the installer document that support 68K- or PPC-based computers. If this field is set to 0, but the PPC Only Document field is non-zero, then the installation plug-in automatically hides this software item when running on a 68K-based computer.
Installer doc ref (PPC only)	The ID of a 'flrf' resource containing the file location of the installer document to use when running on a PowerPC-based computer. If both document references are non-zero, then the installation plug-in automatically selects the document based on the processor type.
Preflight code resource type	The resource type of the preflight code resource.
Preflight code resource ID	The resource ID of the preflight code resource.
Preflight code resource refcon	A 4-byte value passed to the preflight code resource.

Conclusion Plug-in Preference Resource ('ccpr')

```
#define    CCPPrefFlags                                              \
       boolean    kDefaultMessage, kCustomMessageProvide;            \
       fill bit[15]                /* Reserved */

type 'ccpr' {
    switch {
       case format1:
          key integer = 1;         /* Format version */
          CCPPrefFlags;            /* Flags */
          integer;                 /* Plug-in ref when continuing - 'STR ' Rsrc ID */
          integer;                 /* Plug-in string list - 'STR#' Rsrc ID */
    };
};
```

kDefaultMessage / kCustomMessageProvide flag

Use kDefaultMessage to use the built-in restart and quit alert text. Use kCustomMessageProvide to override the default text and use the text provided in the data file.

Plug-in reference when continuing	The ID of a 'STR ' resource containing the name of the plug-in to advance to when the user clicks the Continue button in the conclusion alert. This will usually be the preceding plug-in.
Plug-in string list Rsrc ID	The ID of a 'STR#' resource containing text strings required by the plug-in. 'STR#' resource string index definitions: 1. Custom restart message text 2. Custom quit message text

Writing Upgrader Plug-ins

About the Upgrader Plug-in

Existing Upgrader-based programs can be extended and new Upgrader-based programs can be created by writing new plug-in files. Programmers familiar with writing applications using the Macintosh Toolbox in order to display dialogs and interact with users should feel very comfortable writing new plug-ins. The Upgrader API extends and simplifies the common panel management tasks that most plug-in writers will need to perform.

An Upgrader plug-in is a file that the Upgrader application loads and executes in the order specified by the data file. Most plug-in developers will find that they need to accomplish the following tasks:

1. Design the visual appearance of your plug-in.
2. Define the information to be stored in the data file.
3. Create a new plug-in project or duplicate an existing project.
4. Create the resources specific to your plug-in (making sure to include the plug-in format resource).
5. Write and compile your plug-in code.
6. Write a ModifierTool Editor and resource definition files so others can easily edit your data file resources.

Human Interface Guidelines

Your plug-in will be more easily integrated into existing Upgrader-based programs or mixed with other plug-ins if your panels follow some basic design guidelines. The Upgrader panel-based human interface shares characteristics with other assistant-type programs developed by Apple and others. An assistant gains its ease of use from proper division of a larger task into steps that are manageable by a wide range of audiences.

A few fundamental rules govern the Upgrader-based programs developed so far:

- Use of a non-resizable window that fits all monitors shipped by Apple.
- Panels are divided into three areas: top header contains graphic and panel title, bottom footer contains navigation buttons and content area is specific to the task step.
- Each panel has common navigation buttons, although the Continue button can be renamed to denote the action to be performed.

- If a panel performs an action, then it automatically continues to the next panel. Otherwise, the user should be notified the action failed and possibly be given the option to skip this step.

In addition to deciding how many panels a single plug-in should use, you may need to split an overly complex plug-in into multiple plug-ins. This approach gives the client of an Upgrader-based program that includes your plug-in an easier way to address a wider range of specialized purposes.

Data File Resources

Clients will be able to reuse your plug-in more easily if you carefully divide your resources between the data file and the plug-in file. The easiest way to help make this decision is to place yourself in the client's position.

Most plug-ins store the following information in the data file:

- All file references
- Any text that might need to be tailored to the context of a different Upgrader-based program or a specific use of the original program
- Values that are compared to environmental parameters to make decisions (such as a machine ID or system version)
- Any name of other plug-ins used in branching

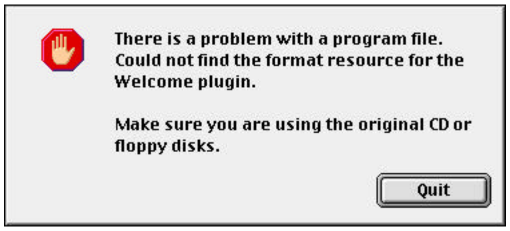
Always supply a resource definition file containing definitions of new resources that you store in the data file. This will be required by clients using the Rez MPW tool. To make the information stored in the data file modifiable by the widest number of clients, we suggest that you create a ModifierTool Editor. As an alternate, supply a template file for use with Resorcerer.

The Plug-in File

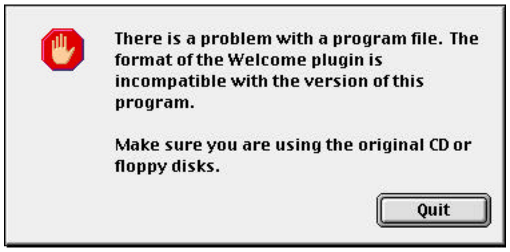
When the Upgrader application locates the plug-in file, it looks for two resources: the plug-in file format resource ('pfmt' ID=1) and the 68K code resource to execute ('PLUG' ID=0). The file will contain other resources required by the plug-in itself.

The Upgrader application uses the plug-in file format resource to ensure the plug-in is compatible with the version of Upgrader before trying to run it. Figure 3-1 shows the alerts you will receive from the Upgrader application upon loading a plug-in file with a missing or incorrect format resource.

Figure 3-1 Alert presented when format of plug-in is not found or is incorrect


There is a problem with a program file. Could not find the format resource for the Welcome plugin.

Make sure you are using the original CD or floppy disks.


There is a problem with a program file. The format of the Welcome plugin is incompatible with the version of this program.

Make sure you are using the original CD or floppy disks.

You can create the format resource using Resorcerer or a resource compiler. Constants in the UpgraderPluginTypes.r file define the version number compatible with the latest version of the Upgrader application.

Upgrader application versions and the plug-in versions they support:

Upgrader Versions	Plug-in File Versions Supported
1.1 - 1.1.1	0.2

Listing 3-1 shows the plug-in file format resource for use with a resource compiler.

Listing 3-1 Plug-in format resource using resource compiler

```
#include "UpgraderPluginTypes.r"

resource ('pfmt', 1) {
   kLatestPluginFileFormatVersion;
}
```

After the plug-in file has been verified as compatible, the Upgrader application looks for the code resource to execute. The current version of the Upgrader application requires that the code resource be created using a development environment that supports A4-based globals. Since Apple's MPW development tools do not, we have adopted the Metrowerks CodeWarrior development environment to present our examples.

The Plug-in Project

Your plug-in project will usually contain everything necessary to build your plug-in, so once it is set up, you can concentrate on writing the code that will implement your panels. While it's easier to start from the example Metrowerks projects provided in the Upgrader SDK, we describe the creation a of new project here in case you do not have access to the SDK.

NOTE

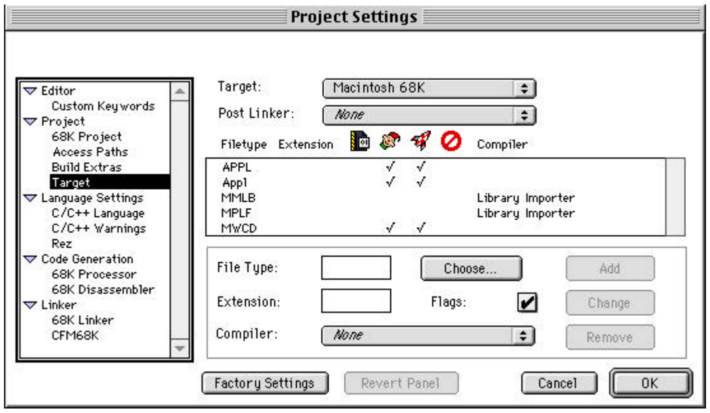
The example projects discussed here are based on release 10 of the Metrowerks CodeWarrior development tools. The latest release will most likely work with minor modifications to your project, but the screen shots provided may look different than the version you are using.

Project Settings

After you have created a new project, choose Project Settings from the Edit menu to display the Project Settings window. Specific settings are required in order to build the plug-in file, so please follow these instructions:

Click "Target" to show the preference panel in Figure 3-2.

Figure 3-2 Target options in the Project Settings window.



Project Settings

- Editor
  - Custom Keywords
- Project
  - 68K Project
  - Access Paths
  - Build Extras
  - Target
- Language Settings
  - C/C++ Language
  - C/C++ Warnings
  - Rez
- Code Generation
  - 68K Processor
  - 68K Disassembler
- Linker
  - 68K Linker
  - CFM68K

Target: Macintosh 68K

Post Linker: None

Filetype	Extension					Compiler
APPL		✓	✓			
Appl		✓	✓			
MMLB						Library Importer
MPLF						Library Importer
MWCD		✓	✓			

File Type:

Extension:

Flags: ✓

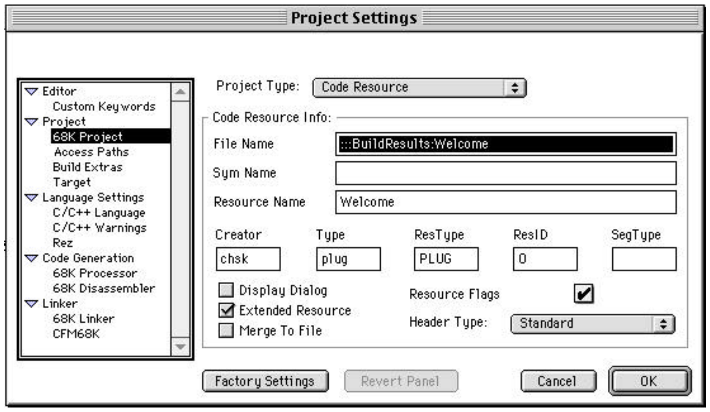
Compiler: None

In the Target preference panel, make these changes:

1. Select Macintosh 68K from the Target pop-up menu

Click “68K Project” to show the preference panel in Figure 3-3.

Figure 3-3 68K Projects options in the Project Settings window.



Project Settings

- Editor
  - Custom Keywords
- Project
  - 68K Project
  - Access Paths
  - Build Extras
  - Target
- Language Settings
  - C/C++ Language
  - C/C++ Warnings
  - Rez
- Code Generation
  - 68K Processor
  - 68K Disassembler
- Linker
  - 68K Linker
  - CFM68K

Project Type: Code Resource

Code Resource Info:

File Name: :::BuildResults:Welcome

Sym Name:

Resource Name: Welcome

Creator	Type	ResType	ResID	SegType
chsk	plug	PLUG	0	

☐ Display Dialog

☑ Extended Resource

☐ Merge To File

Resource Flags: ✓

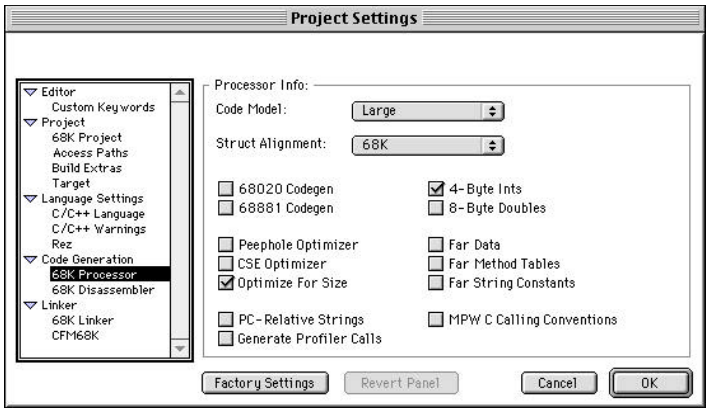
Header Type: Standard

In the 68K Project preference panel, make these changes:

1. Select “Code Resource” from the Project Type pop-up menu .
2. Set Creator to “chsk”.
3. Set Type to “plug”.
4. Set ResType to “PLUG”.
5. Set ResID to “0”.
6. Select Extended Resource option.

Click “68K Processor” to show the preference panel in Figure 3-4.

Figure 3-4 68K Processor options in the Project Settings window



Project Settings

- ▽ Editor
  - Custom Keywords
- ▽ Project
  - 68K Project
  - Access Paths
  - Build Extras
  - Target
- ▽ Language Settings
  - C/C++ Language
  - C/C++ Warnings
  - Rez
- ▽ Code Generation
  - **68K Processor**
  - 68K Disassembler
- ▽ Linker
  - 68K Linker
  - CFM68K

Processor Info:

Code Model: Large

Struct Alignment: 68K

☐ 68020 Codegen	☑ 4-Byte Ints
☐ 68881 Codegen	☐ 8-Byte Doubles
☐ Peephole Optimizer	☐ Far Data
☐ CSE Optimizer	☐ Far Method Tables
☑ Optimize For Size	☐ Far String Constants
☐ PC-Relative Strings	☐ MPW C Calling Conventions
☐ Generate Profiler Calls	

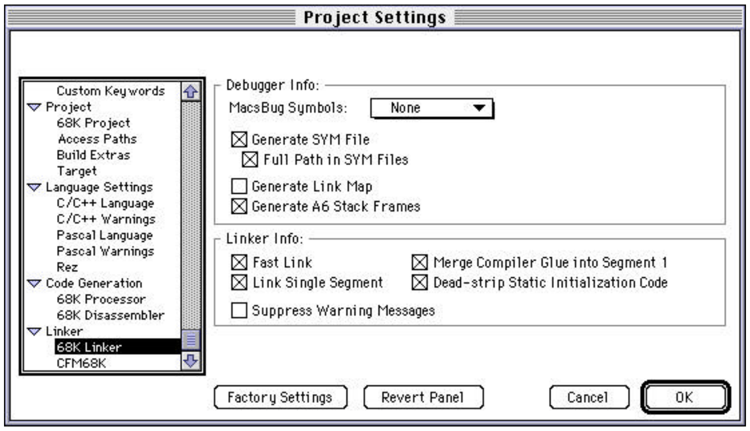
Factory Settings | Revert Panel | Cancel | OK

In the 68K Processor preference panel, make these changes:

1. Select Large from the Code Model pop-up menu.
2. Select the 4-Byte Ints option.

Click “68K Linker” to show the preference panel in Figure 3-5.

Figure 3-5 68K Linker options in the Project Settings window



Project Settings

- Custom Keywords
- ▽ Project
  - 68K Project
  - Access Paths
  - Build Extras
  - Target
- ▽ Language Settings
  - C/C++ Language
  - C/C++ Warnings
  - Pascal Language
  - Pascal Warnings
  - Rez
- ▽ Code Generation
  - 68K Processor
  - 68K Disassembler
- ▽ Linker
  - **68K Linker**
  - CFM68K

Debugger Info:

MacsBug Symbols: None

☒ Generate SYM File
 ☒ Full Path in SYM Files
☐ Generate Link Map
☒ Generate A6 Stack Frames

Linker Info:

☒ Fast Link	☒ Merge Compiler Glue into Segment 1
☒ Link Single Segment	☒ Dead-strip Static Initialization Code
☐ Suppress Warning Messages	

Factory Settings | Revert Panel | Cancel | OK

In the 68K Linker preference panel, make these changes:

1. Select the Link Single Segment option.

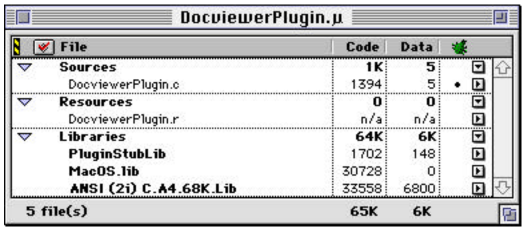
You may also need to add additional paths in the Access Path preference panel depending on where you place the interface and library files provided in the SDK.

Project Files

A new project starts empty, so you’ll need to add files to the project. For a small plug-in project, you might need a single code file, a single resource file, and three or more library files.

Figure 3-6 shows the project window of a small plug-in project

Figure 3-6 Project window



DocviewerPlugin.µ

File	Code	Data	
Sources	**1K**	**5**	
DocviewerPlugin.c	1394	5	•
Resources	**0**	**0**	
DocviewerPlugin.r	n/a	n/a	
Libraries	**64K**	**6K**	
PluginStubLib	1702	148	
MacOS.lib	30728	0	
ANSI (2i) C.A4.68K.Lib	33558	6800	
5 file(s)	**65K**	**6K**	

The files have been grouped into three segments to make organization easier: a segment for the source files, a segment for the resource files, and a segment for libraries.

The source files can be C-based, C++-based or a combination. If you allocate objects in C++ using the new operator, please read the section "Memory Issues". You'll need to include the file "UpgraderPlugin.h" in order to call Upgrader-provided routines.

The resource files can be either ".r" files that must be compiled when your project is built or precompiled resource files that will be copied directly into the plug-in file. If you use ".r" files, then you'll need to include the file "UpgraderPluginTypes.r" from within your ".r" file to have access to the plug-in format resource definition. If you use Resorcerer to create your resource file then you'll want to use the private template file provided in the SDK.

NOTE

Resources stored in the plug-in file should have IDs between 2,000 and 10,000 to avoid conflicts with IDs of resources of the same type stored in the data file.

The library files will be a combination of libraries that come with the Metrowerks development kit and the "PluginStubLib" library file provided in the Upgrader SDK. Make sure to use the libraries that support A4-based globals when given a choice.

Files provided in the SDK that must be included in your project:

File Name	**File Description**
UpgraderPlugin.h	A header file to be included by your ".c" or ".cp" files to have access to the Upgrader API routines, error definitions and various enumerations, constants and structures. This file may also be included by ".r" files to gain access to certain constants. The information contained in this file is detailed in the section "Upgrader API Reference".
UpgraderPluginTypes.r	A resource description file to be included by ".r" files which need the plug-in format resource definition.
PluginStubLib	A library file to be added to the project that implements the Upgrader APIs routines.

Memory Management

When the plug-in code resource is loaded and executed, your code is essentially running as if it's part of the Upgrader application. Memory allocations that you make will, by default, be allocated from the Upgrader's heap. Special care must be taken to allow the Upgrader's memory management strategy to coexist with that of your plug-in.

We explore two important areas of memory management to keep in mind while writing your plug-in: ensuring enough memory is available and preventing memory leaks.

Plug-in Memory Allocation

The Upgrader application is shipped with a partition of 750K. This should handle most plug-ins, but you may need to instruct another developer or client using your plug-in to increase the Upgrader application's partition size based on two main factors: memory needs of the plug-in requiring the most heap space, and the size of data file resources preloaded to support ejectable media. The preloaded resources are the responsibility of the person creating the data file, but the memory required by the plug-in while it is running is the responsibility of the plug-in developer. See the section "Partition Size" in chapter 2 for the equation the developer should use to calculate the appropriate partition.

You should calculate the amount of memory your plug-in requires for all modes of operation, then check that this amount of memory is available within your initialization routine. If you don't have enough memory to continue, tell the user to increase the partition size and then call PSQuitShell. As a general rule, you should leave at least 50K free at all times to enable the Upgrader application to handle your API calls. Make sure to add this 50K to your free memory check during you initialization routine. If your plug-in sets more than 1K of global data then add this amount to the other memory requirements of your plug-in.

You should document the total memory requirements of your plug-in so other developers and clients using your plug-in can calculate an appropriate partition size.

Preventing Memory Leaks

When the Upgrader calls your termination routine, you must deallocate any memory that was allocated either directly or indirectly by you during the run of your plug-in. The easiest way to determine if your plug-in is leaking memory during invocation is to use a heap inspector, such as the "HT" command in MacsBug or a separate application like Zone Ranger. If the number of handles and pointers increase after each run of your plug-in, then you know something is not being deallocated and further investigation is required.

One common, and frustrating, memory leak is the memory pools allocated by the `new` operator. If you are allocating objects in C++ using the default new operator, you will need to compile and link with modified versions of the memory allocation routines and explicitly dispose of this memory from within your termination routine. You can find additional information on this subject on the CodeWarrior CD.

Using the Upgrader API

As a plug-in developer, you will implement plug-in-defined routines and call Upgrader-provided routines to present panels and handle interaction with the user.

NOTE

At the present time, the Upgrader API is only callable from 68K code and must be compiled and linked using a development environment that support A4-based globals. Unfortunately, this means that the APIs are not compatible with the standard MPW compiler.

Setting Up Plug-in-Defined Routines

The Upgrader expects three routines to be defined by your plug-in, which the Upgrader application will call at specific times. An initialization routine, `InitializePluginModule`, is assumed to be the entry point into the code resource and should be your main routine. A termination routine, `TerminatePluginModule`, is called whenever the plug-in gives up control to another plug-in, or the Upgrader quits. An event handler, `HandleEventForPluginModule`, is called for each user event.

You'll need to register your event handler and termination routines using the `PSRegisterHandler` routine from within your initialization routine. Listing 3-2 presents an example of how the `InitializePluginModule` routine should begin (the error checking after the call to `PSRegisterHandler` is omitted for clarity).

Listing 3-2 Registering plug-in-defined routines

```
void InitializePluginModule ( void          *inPSTable,
                              SInt32        inRefCon,
                              Boolean       inEnterAtBeginning )
{
#pragma unused ( inRefCon, inEnterAtBeginning)    // these parameters are never used

       EnterPlugin();

       SetupPlugin(inPSTable);

       (void) PSRegisterHandler(kEventHandlerID,
              (UniveralProcPtr)HandleEventForPluginModule);

       (void) PSRegisterHandler(kTerminationHandlerID,
              (UniveralProcPtr) TerminatePluginModule);

       // ... etc.

       ExitPlugin();
}
```

In order to allow the plug-in access to its A4-based global variables, two routines are provided: `EnterPlugin` and `ExitPlugin`. These routines must bracket any code inside a routine called from the Upgrader application or system that accesses a plug-in global variable.

Listing 3-3 shows how to use `EnterPlugin` and `ExitPlugin` routines.

Listing 3-3 Ensuring access to global variables using `EnterPlugin` and `ExitPlugin` routines

```
Boolean       HandleEventForPluginModule ( EventRecord *inEvent )
{
   Boolean    wasHandled;

   EnterPlugin();

   // ... etc.

   ExitPlugin();

   return wasHandled;
}
```

Managing Panels

The term *panels* in Upgrader is taken to mean the various windows that plug-ins display. Help windows are also implemented as panels, but are set up, shown and hidden with a different set of routines. Panels are described in plug-ins as 'DITL' resources in the plug-in (corresponding 'DLOG' resources are not required).

There are two types of Upgrader panel, *global* panels and *custom* panels. The Global panel is the panel which is shared among all plug-ins. The motivation for sharing a common panel is that when a new plug-in is loaded, rather than completely removing the previous panel and drawing a new panel, one panel can be used and the 'DITL' can simply be swapped, making for a faster and cleaner transition between plug-ins. Custom panels are specific to one plug-in and are only displayed while that plug-in is running and then removed when the Upgrader moves on to the next plug-in.

The Global panel is created with a call to `PSSetupNewPanel`, and the custom panel is created with a call to `PSNewCustomPanel`. `PSSetupNewPanel` will check first to see if the Global panel has already been created and if so, only changes the panel's contents. If the Global panel is not currently being displayed, a new Global panel is created. `PSNewCustomPanel` creates a new panel each time it is called.

WARNING

Never use Upgrader routines to manage the content of a window that you have created using `GetNewDialog`, `NewDialog` or any other Dialog Manager routine.

Managing Panel Contents

The following routines are available to plug-in writers for creating panel items; `PSSetPanelItem`, `PSGetPanelItem`, `PSNewStyledStringItem`, `PSNewStyledTextItem` and `PSNewUserItem`. See the section “Using the Document Viewer” for information about the `PSNewDocViewerItem` routine and creating a DocViewer item in a panel.

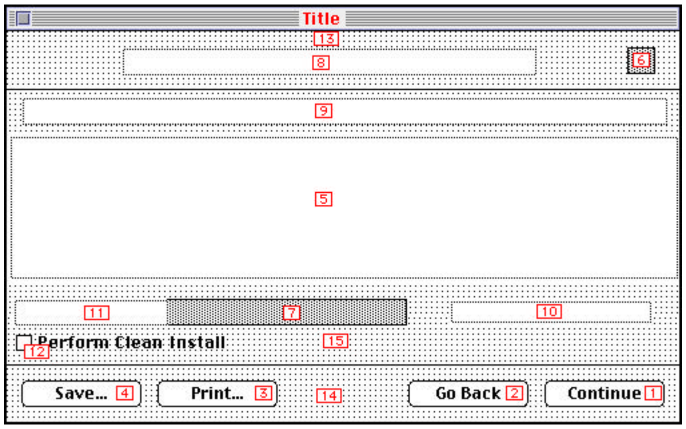
Items which can be placed on the panel include:

- Standard dialog items - All standard dialog items, i.e. anything which can be placed on a 'DITL' can also be used on an Upgrader panel. The two exceptions to this are static text items and edit text items.
- Custom panel items - Upgrader predefined “custom” panel items are created using standard user items and provided routines (see `SetPanelItem`, etc.). The two custom panel items are DocViewer items and static text items. The plug-in writer may also create his / her own custom item using a user item and writing routines to handle it.

Items which are automatically handled by the shell include the default button, the Continue button and the Go Back button (see the description of `PSSetPanelItemAction` for information on how the plug-in writer can instruct the shell to handle these). The Help button is not automatically handled by the shell, the plug-in writer must handle this.

Figure 3-7 shows an example 'DITL' set up with a collection of items.

Figure 3-7 Example 'DITL' defining a panel's contents



Item Number	Item
1	Control item - Continue button (default button)
2	Control item - Go Back button
3	Control item - Print button
4	Control item - Save button
5	User item (used for a DocViewer item)
6	PICT (used for a Help button)
7	Control item - pop-up menu
8	User item - used for static text
9	User item - used for static text
10	User item - depends on value of pop-up menu
11	User item - used for static text
12	Control item - checkbox
13	PICT (used for the title bar)
14	PICT (used for the button bar)
15	PICT (background PICT)

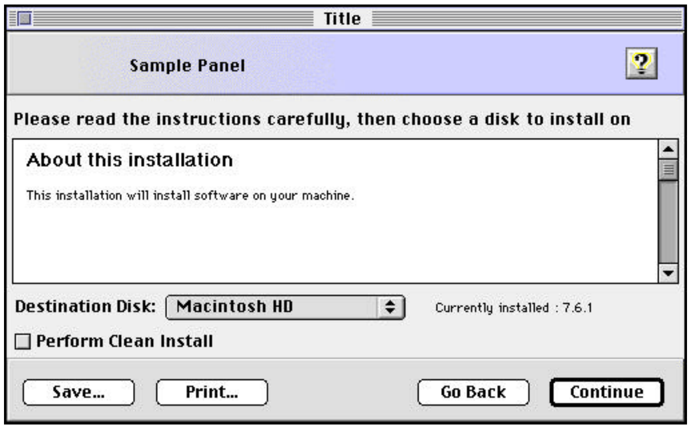
The global panel provided with the shell may have one or a combination of the following targetable panel items:

- User items (e.g. item numbers 5, 8, 9, 10, 11)
- PICTs (e.g. item numbers 6, 13, 14, 15)
- Controls (e.g. item numbers 1, 2, 3, 4, 7, 12)

Note that the shell handles setting the window title. PICTs are handled offscreen in Upgrader, so the plug-in writer has the choice of an item appearing transparently over a picture (e.g. item numbers 6, 8, 9, 10, 11) or opaquely (e.g. item number 5, the DocViewer item). See `PSNewUserItem` below for further information on how this can be done.

Figure 3-8 shows the resultant panel.

Figure 3-8 Example panel as displayed to the user



Please read the instructions carefully, then choose a disk to install on

About this installation

This installation will install software on your machine.

Destination Disk: Macintosh HD

Currently installed : 7.6.1

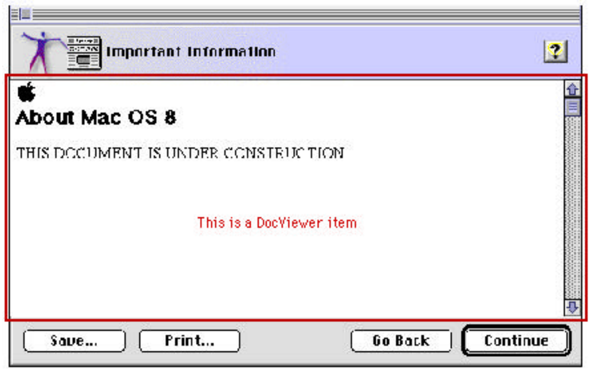
Perform Clean Install

Using the Document Viewer

A DocViewer item is used to view text and/or pictures in an Upgrader panel. The plug-in writer can create a DocViewer item with a user item, and the DocViewer library routines can then be used to handle the item. Upgrader takes care of adding a vertical scroll bar for the DocViewer and resizing the DocViewer item and its contents if the panel is resizable (for example, the help panel supplied with Upgrader has a size box). The shell also handles activate, deactivate and update events for the DocViewer. Plug-in writers simply need to place user items on the panels they are creating and specify the location of any text and/or pictures to be placed in the DocViewer item, via the DocViewer routines detailed below.

Figure 3-9 shows a panel with a DocViewer item.

Figure 3-9 Panel with DocViewer item



About Mac OS 8

THIS DOCUMENT IS UNDER CONSTRUCTION

Four Upgrader routines allow the plug-in writer to interact with the DocViewer item: `PSNewDocViewerItem`, `PSHandleDocScroll`, `PSSaveDoc` and `PSPrintDoc`. `PSNewDocViewerItem` is used to create the a DocViewer item on a panel. `PSSaveDoc` is used to save the DocViewer's contents (if the standard file package is available). `PSPrintDoc` is used to print the DocViewer's contents (if a printer is available). And `PSHandleDocScroll` is used to handle mouse clicks in the DocViewer item, including its scroll bar.

A DocViewer item expects to find a 'STR#' resource of ID 1000 in the plug-in file. See Listing 3-4 for an example of this resource.

Listing 3-4 'STR#' resource required in the plug-in file when using DocViewer items

```
resource 'STR#' (1000) {
       {
          $"CA",            // non-breaking space, used as the PICT delimiters
          "",               // (unused but must be present)
          "untitled",       // default "Save As" field in standard Save dialog
       }
};
```

Listing 3-5 presents the code for setting up a DocViewer item (error-checking is omitted for clarity):

Listing 3-5 Code for setting up a DocViewer item

```
ShellErr      theErr = noErr;
DocInfoHandle docViewerHandle;
DocViewerType docType;
Handle        itemHandle;
Rect          itemRect;
PanelItemType itemType;
short         prefsFlags, prefsTextID, prefsPICTBaseResID;

/*
   do first:
   read prefsFlags, prefsTextID, prefsBasePICTResID from wherever
   they should be read (e.g. plug-in resource fork/data file)
*/

(void) PSGetPanelItem(gMyPluginPanel, kDocViewerItemNumber,
              &itemType, &itemHandle, &itemRect);

if ((prefsFlags & kMainTextInFile) == kMainTextInFile)
   docType = kDocFileType;
else
   docType = kDocResType;

docViewerHandle = PSNewDocViewerItem(gMyPluginPanel, &itemRect,
              docType, prefsTextID, prefsPICTBaseResID);
if (docViewerHandle != NULL)
   (void) PSSetPanelItem(gMyPluginPanel, kDocViewerItemNumber,
              docType, (Handle) docViewerHandle, &itemRect);
else
   theErr = kCannotLoadNeededResourceErr;
```

Listing 3-6 shows code for handling scrolling in a DocViewer item. This is generally called from `HandleEventForPluginModule` (error-checking is omitted for clarity):

Listing 3-6 Code for handling DocViewer item events

```
WindowPtr     whichWindow;
short         windowPart;
Point         localPt = inEvent->where;
PanelItemType itemType;
Handle        itemHandle;
Rect          itemRect;
Boolean       wasHandled = false;

windowPart = FindWindow(localPt, &whichWindow);
if (whichWindow == gMyPluginPanel) {
   switch (windowPart) {
      case inContent:
         PSGetPanelItem(gMyPluginPanel, kDocViewerItem, &itemType, &itemHandle, &itemRect);

         GlobalToLocal(&localPt);
         if (PtInRect(localPt, &itemRect)) {
            (void) PSHandleDocScroll(gMyPluginPanel, localPt, kDocViewerItem);
            wasHandled = true;
         }
         else {
            /* do:
            deal with mouseclicks elsewhere on the panel
            */
         }
      break;

      // ... etc.
   }
}
```

Navigation

Upgrader will usually have several plug-ins; to navigate between them Upgrader provides routines that allow the user to proceed to the next or return to the previous plug-in. The default plug-in sequence is defined in the sequence resource. See the description of the `PSGoToNextPlugin` routine for information on how this default sequence may be overridden.

Upgrader uses an internal plug-in history stack to determine the previous plug-in. This stack ensures the plug-in always returns to the most recently visited plug-in. This stack-based approach to determining the previous plug-in is used instead of simply examining the sequence resource and finding the default previous plug-in, since in the case where Upgrader has overridden the default plug-in sequence, the default previous plug-in will not be the one most recently visited.

NOTE

Normally when the user uses `PSGoToNextPlugin` to proceed to the next plug-in the details of the new plug-in are added in to the internal Upgrader plug-in history stack. But in the case where the plug-in that Upgrader is moving on to is one which has already been visited (i.e. Upgrader finds an earlier reference to the plug-in in the history stack), Upgrader will instead use the reference to the plug-in already contained in the history stack and discard all references to plug-ins after this point in the history stack.

Using this method is less disorientating to the Upgrader user than the simpler alternative where the history stack could grow to a large size if the default plug-in sequence is overridden a number of times.

Here’s an example to help clarify how the history list works. The following is the default plug-in sequence:

1 -> 2 -> 3 -> 4 -> 5

For this example plug-in 4 will not always be a necessary step, it could be a license agreement that plug-in 3 could decide doesn’t need to be shown to the user.

If plug-in 4 is in fact skipped and Upgrader moves straight on to plug-in 5, the history stack would look like the following:

1 -> 2 -> 3 -> 5

So in this example if `PSGoToPreviousPlugin` is called from plug-in 5, Upgrader will return to plug-in 3.

Following on with the same example, if plug-in 5 has an option where it is necessary to go to another plug-in, for example if the plug-in wants the user to go back to plug-in 2 to perhaps reset the destination disk. In this case the stack will *not* grow as follows:

1 -> 2 -> 3 -> 5 -> 2 (will not happen)

What will occur, is the stack gets unrolled back to the first instance of plug-in 2 and so the stack will simply look as follows:

1 -> 2

NOTE

There will be cases where the plug-in writer does not want a plug-in added to the plug-in history list since it may only need to be run once. An example would be a plug-in which checks that the machine on which Upgrader is running is capable of running the software that is to be installed. To indicate to Upgrader that a plug-in is of this type, use the `onlyRunOnce` plug-in flag when filling-in details of the plug-in to the sequence resource.

Managing the Help Window

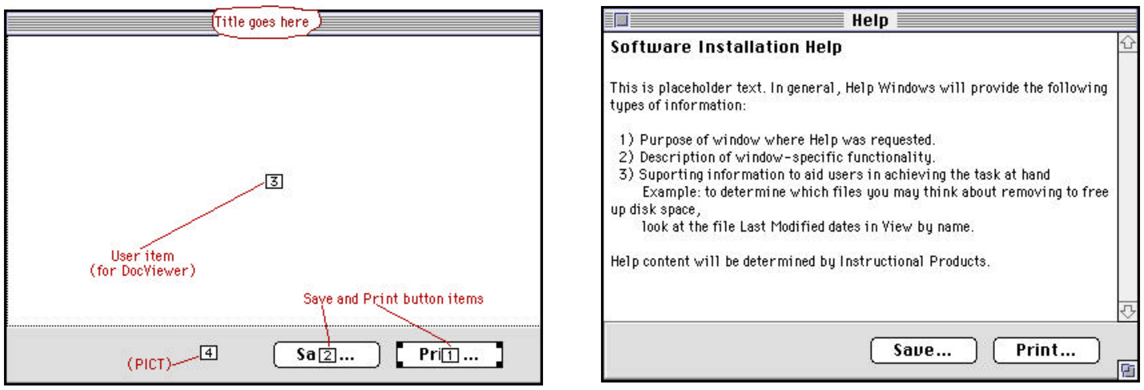
Support for help panels is supplied in Upgrader to allow for the easy inclusion of simple, standard help windows in a plug-in. The help API calls provided with Upgrader do not have to be used. Similarly, the default help panel ('DLOG'/'DITL' ID of 1050) provided with the Shell doesn’t have to be used, however if plug-in writers wish to design their own panels, the following item numbers must be used:

```
enum {
       kHelpPrintButton = 1,        // Print Button item number
       kHelpSaveButton,             // Save Button item number
       kHelpDocItem                 // User item number for DocViewer
};
```

The help panel provided with the shell has the following targetable panel items:

- User item (for the DocViewer item, i.e. the help panel contents)
- Print button (see `PSCheckEnvironment` for more details)
- Save button (see `PSCheckEnvironment` for more details)

Figure 3-10 Help window in Resorcerer and as shown to the user



Software Installation Help

This is placeholder text. In general, Help Windows will provide the following types of information:

1) Purpose of window where Help was requested.
2) Description of window-specific functionality.
3) Suporting information to aid users in achieving the task at hand
Example: to determine which files you may think about removing to free up disk space,
look at the file Last Modified dates in View by name.

Help content will be determined by Instructional Products.

See the section “Using the Document Viewer” for further information on where DocViewer items are read from, stored, etc.

Four help routines are available to plug-in writers to setup the help window, display it, close it and handle events for it: `PSSetupHelpWindow`, `PSDisplayHelpWindow`, `PSCloseHelpWindow`, and `PSHandleHelpWindowEvent`.

Listing 3-7 demonstrates calling `PSHandleHelpWindowEvent` from inside `HandleEventForPluginModule` (error-checking is omitted for clarity):

Listing 3-7 Example code for handling help window events

```
Boolean HandleEventForPluginModule( EventRecord *inEvent )
{
   Boolean    wasHandled = false;
   WindowPtr  theWindow;

   EnterPlugin();

   switch (inEvent->what) {
      /* do:
        handle any other plug-in-specific cases in here
      */

      case mouseDown:
         (void) FindWindow(inEvent->where, &theWindow);
         if (theWindow == gMyPluginHelpPanel)
            wasHandled = PSHandleHelpWindowEvent(gMyPluginHelpPanel, inEvent);
      break;

      case osEvt:
         // --- this deals with resume events only
         if(((unsigned long) inEvent->message >> 24) == suspendResumeMessage){
            if ((inEvent->message & resumeFlag) != 0) {
               // --- if there is a print button on the current panel, use this code
               ControlHandle         printButtonControl;
               EnvironmentType   environmentFlags;
               short                 printButtonHilite = 255;
               GrafPtr               savedPort;
               PanelItemType         userItemType;
               Rect                  itemRect;

               GetPort(&savedPort);
               SetPort(gMyPluginPanel);
```

```
                    (void) PSCheckEnvironment(&environmentFlags);
                    (void) PSGetPanelItem(gMyPluginPanel,kPrintButtonItem,
                                      &itemType, (Handle *) &printButtonControl, &itemRect);

                    if ((environmentFlags & kPrinterAvailableMask) == 0)
                        printButtonHilite = 255;
                    else
                        printButtonHilite = 0;

                    if (FrontWindow() == gMyPluginPanel)
                        HiliteControl((ControlHandle) printButtonControl, printButtonHilite);

                    // --- Note: this line should not be removed
                    SetCRefCon((ControlHandle) printButtonControl, printButtonHilite);

                    SetPort(savedPort);

                    // --- let the Shell have a go at the standard help panel resume events
                    (void) PSHandleHelpWindowEvent(gMyPluginHelpPanel, inEvent);
                }
            }
        break;

        default:
        break;
    }

    ExitPlugin();

    return wasHandled;
}
```

Exchanging Data with other Plug-ins

Plug-ins can use the Global Data routines to store (`PSSetGlobalData`) and retrieve (`PSGetGlobalData`) data that persist between invocations of a plug-in. Publicly defined global data types allow plug-ins to communicate with one another. A plug-in's documentation should describe which global data types it reads and which it sets. Privately defined global data types allow a single plug-in to store information until its next invocation.

NOTE

Don't confuse global data with global variables. Global variables are identifiers you define within your code that have global scope but are only valid while the plug-in is running and disappear when the plug-in terminates.

Referencing Files

If a plug-in needs to access a file other than the data file or plug-in file, it should use a file reference resource ('flrf') to store the path to this file. Upgrader has support for finding a file based on its file reference resource ID. Since data should generally be stored in the data file, the most common way for a plug-in to find a file directly is to launch an application, with or without a document. Upgrader API provides the `PSLaunchFile` routine to make it easy for plug-ins to launch an application specified by a file reference resource ID.

See the section "Upgrader Application Reference" for a detailed description of the file reference resource.

Displaying Alerts

The Upgrader API provides two routines (`PSAlert` and `PSErrorAlert`) for displaying auto-sized dialogs for the purpose of displaying errors or other simple alerts containing a text message. Common alert dialogs are provided in the Upgrader, but the plug-in writer can define new dialogs and store these within the plug-in file.

The `PSAlert` routine is used to display an alert when you have a text string to display and the ID of a 'DLOG' resource to display the text within. The 'DLOG' resource can be one the predefined dialogs contained in the Upgrader application, or one that you define in your plug-in file.

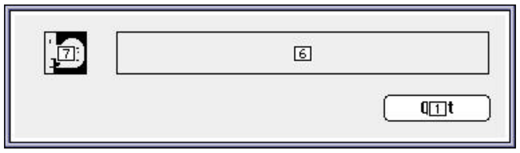
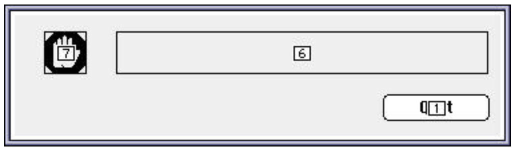
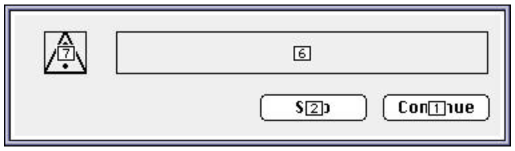
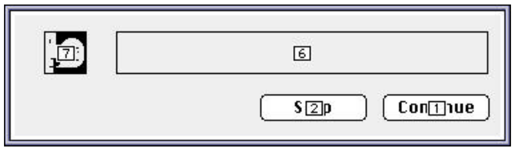
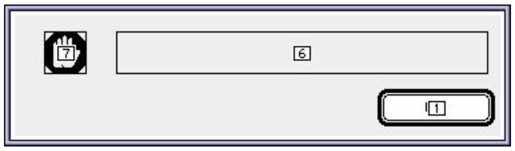
The `PSErrorAlert` routine is used in conjunction with a 'ners' resource to look up the text message and dialog based on either a Upgrader-defined error number or a error number defined by your plug-in.

Displaying Upgrader-Defined Error Alerts

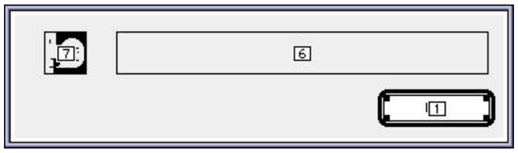
You can call the Upgrader routine `PSErrorAlert` to display one of the Upgrader-defined error numbers that you might have received as the result of calling a Upgrader routine. See the section “Summary of the Upgrader API” for a listing of the Upgrader-defined error numbers available for use with the `PSErrorAlert` routine.

Table 3-1 shows the dialogs supplied with the Upgrader application to be used with the `PSAlert` routine and when defining your own error numbers to be used with the `PSErrorAlert` routine.

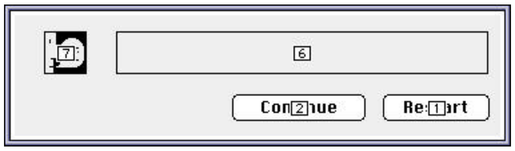
Table 3-1 Dialogs supplied in the Upgrader application

Dialog ID	**Dialog Layout**
kStopDLOGID (600)	
kStopAndQuitDLOGID (526)	
kSkipOrContinueDLOGID (520)	
kStopOrContinueDLOGID (506)	
kStopOKDLOGID (519)	

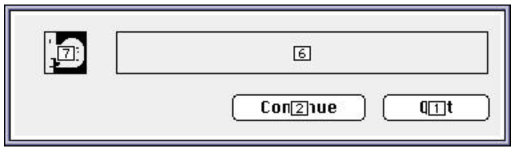
kNoteOKDLOGID (508)



kRestartOrContinueDLOGID (514)



kQuitOrContinueDLOGID (507)



Displaying Plug-in-Defined Errors

The 'ners' resource connects the message to be displayed with the dialog to be shown by referencing the 'DLOG' resource ID and the index within a 'STR#' resource. When a plug-in writer wants to display messages that are specific to a plug-in this is done by creating a 'ners' resource ID 500 in the plug-in file, and also creating a 'STR#', containing the error messages, with ID 520. The individual messages are then added to the 'STR#' resource, making sure that they correspond to the indices for each error message number in the 'ners' resource.

NOTE

'ners' resources must always use the resource ID 500 and the 'STR#' reference from a 'ners' resource must always have a resource ID of 520.

If none of the 'DLOG's available in the shell are suitable, a 'DLOG' resource can also be added to the plug-in file. See `PSAlert` and `PSErrorAlert` for more information.

Listing 3-8 shows an example 'ners' resource as it might appear in a ".r" file, which connects the error number defined by the plug-in with the message and dialog to be displayed.

Listing 3-8 An example 'ners' 500 resource used in conjunction with a 'STR#' 520 resource

```
resource 'ners' (500, nonpurgeable) {
   {
      1000,   1, kStopDLOGID,
      1001,   2, kStopDLOGID,
   }
};

resource 'STR#' (520, nonpurgeable) {
   {
      "Unable to determine if your computer is compatible with this program.";
      "This program cannot run on your computer.";
   }
};
```

Supporting Multiple Source Disks

The main motivation for preloading resources is to eliminate extra disk swaps when the user is installing from multiple disks. Upgrader overrides the normal method of preloading resources (by selecting the preload attribute for the necessary resource) to ensure that all the various resources from all the Upgrader files are preloaded when necessary. There are a number of situations when plug-in resource preloading should be considered:

- If Upgrader is launched from a server and there is a possibility of the connection to the server being lost. If this happens, Upgrader will display an alert and then quit, calling the current plug-in's termination handler. The resources required to display this error message will already be preloaded but it is possible that resources will also be required by the plug-in, possibly to update a panel or for use in the plug-in's termination routine.
- If Upgrader is running from floppy disks or some other ejectable media and there is a possibility of the plug-in requiring a file on a different floppy (this is most likely to be the case if the plug-in uses `PSLaunchFile` to open another application). There is also the possibility that the data file and the plug-in are on different floppy disks.

Since plug-ins usually have some of their resources in the data file and some in the plug-in file itself, plug-in writers must ensure that both sets of resources in both are preloaded when necessary.

- For plug-in resources in the data file, each plug-in should have a 'RES#' resource in the data file that contains the list of all the resources in the data file relevant to the plug-in. Upgrader will find this list using the “Data file preload list” element in the plug-in's section of the 'tsqc' resource. The Upgrader will only preload these resources if it determines that the plug-in's volume is different volume to the data file's.
- For plug-in resources in the plug-in file that the plug-in writer decides are necessary to preload, another 'RES#' resource must be created (and placed in the plug-in file). The plug-in itself will have to determine if it is necessary to preload these and if so preload them using Upgrader routine `PSCollect`.

Listing 3-9 shows a simple example of one as it would look in a .r file:

Listing 3-9 An example 'RES#' resource

```
resource 'RES#' ( 3500, nonpurgeable ) {
    {
        'wppr', 3500,
        'STR#', 3500,
        'TEXT', 3502,
    }
};
```

See the section “Upgrader Application Reference” for a detailed description of the 'RES#' resource.

Upgrader API Reference

Plug-in-Defined Routines

There is one required entry points to the plug-in code resource, `InitializePluginModule` and two other optional entry point routines, `HandleEventForPluginModule` and `TerminatePluginModule`.

InitializePluginModule

```
void InitializePluginModule ( void   *inPSTable,
                              SInt32      inRefCon,
                              Boolean     inEnterAtBeginning );
```

DESCRIPTION

This is the main entry point of the plug-in, initially called by the Shell. This routine must first do some standard initialization and then may perform any initialization tasks that the plug-in itself needs.

PARAMETERS

`inPSTable`	An internal data structure that will be used by `SetupPlugin` to allow the plug-in to access the Upgrader library routines.
`inRefCon`	The RefCon value stored in the sequence resource for this plug-in. The standard use of this parameter is to pass the ID of the preference resource to the plug-in in the low-word so multiple preference resources can be used with a single plug-in file.
`inEnterAtBeginning`	True if the user is entering the plug-in from the previous plug-in.

HandleEventForPluginModule

```
Boolean HandleEventForPluginModule ( EventRecord *inEvent );
```

DESCRIPTION

This routine will be called each time through the Upgrader's event loop. The plug-in will decide whether or not to handle the event passed in itself and handle the event if it's a plug-in event. A boolean is returned by the plug-in to indicate whether or not the event was handled (TRUE if the plug-in handled the event itself).

PARAMETERS

`inEvent`	A pointer to the event which is due for processing

RETURNS

TRUE if the routine (and / or child routines) handled the event and / or no further processing is required by the Shell, FALSE if the event wasn't handled or if the event needs further handling by the Shell.

SPECIAL CONSIDERATIONS

• It is important to set the return value to be TRUE if the event was handled by `HandleEventForPluginModule`, since the Shell handling of the event may be unpredictable if it is being handled a second time.

• For resume events, the plug-in *must* return FALSE to the Shell to allow the Shell a turn at processing the event.

TerminatePluginModule

```
void TerminatePluginModule ( void );
```

DESCRIPTION

The exit point of a plug-in. Performs any tidying-up tasks needed by the plug-in when it quits. This gets called by the Shell when the application quits, when moving to the next plug-in, or when a serious error occurs.

Setup Routines

EnterPlugin

```
EnterPlugin();
```

DESCRIPTION

This routine must be called on entry to any routine in the plug-in that may be called directly from the Upgrader, such as initial entry point, termination entry point, event entry point and any callback routines, including both System callbacks and Upgrader callbacks (e.g. user item draw procedures, etc.). It allows the plug-in to access its global variables.

SPECIAL CONSIDERATIONS

• This macro saves-off the current A4 world.

SEE ALSO

CodeWarrior documentation for more information on `EnterCodeResource`.

ExitPlugin

```
ExitPlugin();
```

DESCRIPTION

This routine must be called just before exiting any routine that may be called directly from the Shell, such as initial entry point, termination entry point, event entry point and any callback routines, including both System callbacks and Upgrader callbacks (e.g. user item draw procedures, etc.). It removes access to the plug-in’s global variables. A call to `ExitPlugin` should match a previous call to `EnterPlugin` in the same routine.

SPECIAL CONSIDERATIONS

• This macro restores the current A4 world.

SEE ALSO

CodeWarrior documentation for more information on `ExitCodeResource`.

NOTE

It is important that before `EnterPlugin` is called and after `ExitPlugin` is called, the routine must not try to access global variables. One place where it is easy to use global variables incorrectly is if a `Str255` is declared in the variable declaration section.

CodeWarrior stores strings as globals. Another potential error would be to try returning the value of a global from a routine after the call to `ExitPlugin` has been made. See code listings 3-10 and 3-11 for an illustration of this.

Listing 3-10 shows an example of the incorrect use of `EnterPlugin` and `ExitPlugin`:

Listing 3-10 Incorrect use of the `EnterPlugin` routine

```
SInt16          gSomeGlobalVariable;

Boolean         HandleEventForPluginModule ( EventRecord *inEvent )
{
        Str255          myString = "\pThis is my string";
        Boolean         wasHandled = true;

        gSomeGlobalVariable = 0;

        EnterPlugin();

        // ... etc.

        ExitPlugin();

        return wasHandled;
}
```

Listing 3-11 shows an example of the correct use of `EnterPlugin` and `ExitPlugin`:

Listing 3-11 Correct use of the `EnterPlugin` routine

```
SInt16          gSomeGlobalVariable;

Boolean         HandleEventForPluginModule ( EventRecord *inEvent )
{
        Str255          myString;
        Boolean         wasHandled = true;

        EnterPlugin();

        MyPStringCopy(myString, "\pThis is my string");
        gSomeGlobalVariable = 0;

        // ... etc.

        ExitPlugin();

        return wasHandled;
}
```

SetupPlugin

The `SetupPlugin` routine is primary used in plug-ins is to allow access to the Upgrader routines. It also allows access to the Upgrader's QuickDraw globals which are needed for any plug-in drawing procedures. This routine needs only to be called once and this will always be from the `InitializePluginModule` routine.

```
void SetupPlugin ( void *shellFunctions );
```

DESCRIPTION

Performs several internal initialization routines for the plug-ins, one of which sets up an internal table of all the Upgrader routines available to the Shell. So, this routine must be called by the plug-in in `InitializePluginModule` before any other Upgrader routines are called.

PARAMETERS

`shellFunctions`	A table of all the routines available to the plug-in, the structure of this is not available to the plug-ins.

SPECIAL CONSIDERATIONS

• This must be called before any other Upgrader routines are called.

• This routine must be preceded with a call to `EnterPlugin` and ultimately succeeded by a call to `ExitPlugin`.

PSRegisterHandler

```
ShellErr PSRegisterHandler ( HandlerIDType      inHandlerType,
                             UniversalProcPtr   inHandlerProcPtr );
```

DESCRIPTION

Gives the Upgrader the necessary access to the plug-in’s `HandleEventForPluginModule` and `TerminatePluginModule` plug-in entry point routines.

PARAMETERS

`inHandlerType`	`kEventHandlerID` if registering the `HandleEventForPluginModule` routine, or `kTerminationHandlerID` if registering the `TerminatePluginModule` routine
`inHandlerProcPtr`	A pointer to the plug-in’s `HandleEventForPluginModule` or `TerminatePluginModule` routine

RETURNS

`noErr`	The routine was successfully registered with the Shell
`kUnknownPluginHandlerErr`	`inHandlerType` was not recognized or handled by the Shell

SPECIAL CONSIDERATIONS

• Support for UniversalProcPtrs for PPC code is not provided in Upgrader.

NOTE

It is important to call `SetupPlugin` before `PSRegisterHandler` is called, as the plug-in won’t have any access to the Upgrader routines including `PSRegisterHandler`, until after the call to `SetupPlugin`.

Panel Handling Routines

Several routines are available to the plug-in writer to handle Upgrader panels and these are now detailed.

PSSetupNewPanel

```
ShellErr PSSetupNewPanel ( SInt16   inPanelItemsRsrcID,
                           PanelPtr  *outPanelPtr );
```

DESCRIPTION

This is used to setup the contents of a global panel based on the contents of the specified 'DITL'. If the global panel is not already open then it creates the panel, if the panel is already open it changes the contents of the panel.

PARAMETERS

`inPanelItemsRsrcID`	A 'DITL' resource ID which specifies the contents of the panel
`outPanelPtr`	If successful, returns a `PanelPtr` to the panel. On failure (panel couldn’t be opened or changed), returns NULL.

RETURNS

`noErr`	The panel was correctly set up
`kInternalErr`	If the global `PanelPtr` is NULL and we are attempting to change its contents
`MemError`	Standard memory error
`ResError`	Standard resource error

SPECIAL CONSIDERATIONS

• This routine does not display the panel or its new contents, call `PSShowPanel` to do this.

PSNewCustomPanel

```
ShellErr PSNewCustomPanel ( short    inPanelItemsRsrcID,
                            DVFlags   inFlags,
                            PanelPtr  *outPanelPtr );
```

DESCRIPTION

Sets up the contents of a new custom panel based on the 'DITL' specified. This routine will always create a new window for the panel.

PARAMETERS

`inPanelItemsRsrcID`	A 'DITL' resource ID which specifies the contents of the panel
`inFlags`	Always `kGrowWindow`, which causes `PSNewCustomPanel` to load a resizable window (with a Grow Box)
`outPanelPtr`	On success, returns a `PanelPtr` to the panel. On failure (panel couldn’t be opened or changed), returns NULL.

RETURNS

`noErr`	The new custom panel was correctly set up
`kInternalErr`	If the global `PanelPtr` is NULL and we are attempting to change its contents
`MemError`	Standard memory error
`ResError`	Standard resource error

SPECIAL CONSIDERATIONS

• This routine does not display the panel or its new contents, call `PSShowPanel` to do this.

• The Shell automatically handles the growing of panels, i.e., it will move buttons, resize DocViewers, etc.

PSUpdatePanel

```
ShellErr PSUpdatePanel ( PanelPtr inPanelPtr );
```

DESCRIPTION

Redraws the specified panel. This is normally called in response to an update event for the current panel. Most plug-ins will not need to directly call this routine and should return FALSE in `HandleEventForPluginModule`, thereby forcing the Shell to handle the update event. In the event that is is needed to be explicitly called, it should be bracketed with calls to `BeginUpdate` and `EndUpdate` in the calling routine.

PARAMETERS

`inPanelPtr`	A pointer to the panel which is to be drawn

RETURNS

`noErr`	The panel was drawn correctly
`paramErr`	An illegal parameter was passed to `NewGWorld`
`cDepthErr`	An invalid pixel depth was passed to `NewGWorld`
`QDErr`	Standard QuickDraw error

SPECIAL CONSIDERATIONS

• This routine assumes that suitable bracketing with `BeginUpdate` and `EndUpdate` calls is provided in the calling routine.

• It is the responsibility of the calling routine to erase the regions of the window that need to be redrawn.

SEE ALSO

Inside Macintosh: Imaging with QuickDraw 6-16 for more information on `NewGWorld`.

PSActivatePanel

```
ShellErr PSActivatePanel ( PanelPtr  inPanelPtr,
                           Boolean    inShouldActivate );
```

DESCRIPTION

This routine is used to activate or deactivate panels, depending on the value of `inShouldActivate`. Most plug-ins will not need to directly call this routine and should return FALSE in `HandleEventForPluginModule`, thereby forcing the Shell to handle the activate / deactivate event. Typically, a plug-in might call this directly before displaying a dialog box to deactivate the panel and then to reactivate the panel when the dialog box is closed.

PARAMETERS

`inPanelPtr`	A pointer to the panel to be activated
`inShouldActivate`	If TRUE, activate the panel, if FALSE, deactivate the panel

RETURNS

`noErr`	Always returns `noErr`

SPECIAL CONSIDERATIONS

• This routine sets the current port to the panel window.

PSDisposePanel

```
void PSDisposePanel ( PanelPtr inPanelPtr );
```

DESCRIPTION

Removes a panel from the screen, disposes of its window and releases the memory occupied by all structures associated with the panel.

PARAMETERS

`inPanelPtr`	A pointer to the panel to be disposed of

SPECIAL CONSIDERATIONS

• There is no need to explicitly dispose of / hide the global panel if the next plug-in is going to change its contents or hide it.

• This should not be called by a plug-in to close the global panel, `PSHidePanel` should be called instead.

• This should only be used in conjunction with panels created with `PSNewCustomPanel`.

PSShowPanel

```
ShellErr PSShowPanel ( PanelPtr inPanelPtr );
```

DESCRIPTION

Shows the panel if it was hidden, else brings it to the front.

PARAMETERS

`inPanelPtr`	A pointer to the panel to be shown

RETURNS

`noErr`	Always returns `noErr`

PSHidePanel

```
void PSHidePanel ( PanelPtr inPanelPtr );
```

DESCRIPTION

Hides the specified panel.

PARAMETERS

`inPanelPtr`	A pointer to the panel to be hidden

PSGetPanelItemHit

```
Boolean PSGetPanelItemHit (   PanelPtr         inPanelPtr,
                              EventRecord      *inPanelEvent,
                              short            *outItemHit );
```

DESCRIPTION

Call this routine in response to a mousedown event in a panel to find which item on the panel was hit by the user.

PARAMETERS

`inPanelPtr`	A pointer to the panel in which the mousedown occurred
`inPanelEvent`	The event record for the mousedown event
`outItemHit`	On return, the item number of the item that was hit on the panel (if an item on the current panel was clicked), else `outItemHit` is undefined

RETURNS

TRUE if an active control item on this panel was clicked, FALSE otherwise.

PSSetPRefCon

```
void PSSetPRefCon (     PanelPtr    inPanelPtr,
                        long        inRefCon );
```

DESCRIPTION

Sets the panel’s refCon.

PARAMETERS

`inPanelPtr`	A pointer to the panel whose refCon is to be set
`inRefCon`	A long value which can be used by the plug-in for its own use

SPECIAL CONSIDERATIONS

• This is needed since the panel module uses the standard window’s refCon to store information.

PSGetPRefCon

```
long PSGetPRefCon ( PanelPtr inPanelPtr );
```

DESCRIPTION

Gets the panel’s refCon.

PARAMETERS

`inPanelPtr`	A pointer to the panel whose refCon we wish to retrieve

RETURNS

The value of the refCon stored in the panel record’s refCon field, or 0 (zero) if `inPanelPtr` is not a valid panel pointer.

SPECIAL CONSIDERATIONS

• This is needed since the panel module uses the standard window’s refCon to store information.

PSSetPanelItemAction

```
ShellErr PSSetPanelItemAction ( PanelPtr         inPanelPtr,
                                short            inItemNumber,
                                PanelActionType  inActionType );
```

DESCRIPTION

Sets the “action” attributes for a specified panel item. These attributes are drawing the default button and handling the Continue, Go Back, or Quit buttons. The main motivations for using this routine are ease of plug-in development and the standardization of plug-in behavior. Constants for specifying panel item actions are defined in the “UpgraderPlugins.h” file.

The following details the keys mapping:

Key	**Mapped To**
Return	Default button
Enter	Default button
Escape	Go Back button
Clear	Go Back button

PARAMETERS

`inPanelPtr`	A pointer to the panel containing the item whose action attributes are to be set
`inItemNumber`	The panel item number of the item whose action attributes are to be set
`inActionType`	The action attributes to be set for the specified item

RETURNS

`noErr`	The action attributes were set for the specified item
`kItemTypeMismatchErr`	The specified item was not a control item
`kPanelItemNotFoundInListErr`	The specified item was not found on the panel

Example, setting panel button actions:

```
// setting the default button:
   (void) PSSetPanelItemAction(gMyPluginPanel, kContinueButtonItem, kDefaultButtonMask);

// setting the Continue button:
   (void) PSSetPanelItemAction(gMyPluginPanel, kContinueButtonItem, kContinueButtonMask);

// setting the Go Back button:
   (void) PSSetPanelItemAction(gMyPluginPanel, kGoBackButtonItem, kGoBackButtonMask);
```

Panel Content Routines

Following are the routines for handling Upgrader panels.

PSSetPanelItem

```
ShellErr PSSetPanelItem ( PanelPtr        inPanelPtr,
                          short           inItemNumber,
                          PanelItemType   inItemType,
                          Handle          inItemHandle,
                          Rect            *inItemRect );
```

DESCRIPTION

This routine works like `SetDialogItem`. It can be used to change the type or data of a specified panel item, or can be used to move the item on the panel (by changing its boundary rectangles).

PARAMETERS

`inPanelPtr`	A pointer to the panel containing the item to change
`inItemNumber`	The panel item number of the item to change
`inItemType`	A value which represents the type of item in the `itemNumber` parameter.
`inItemHandle`	A pointer to the draw procedure for a user item, or a handle to the item to be changed for all other items
`inItemRect`	The display rectangle (in the panel's local coordinates) of the item to be changed

RETURNS

`noErr`	The item was changed successfully
`kCantChangePanelItemToSpecifiedTypeErr`	The item couldn't be changed

SPECIAL CONSIDERATIONS

• This routine disposes of the previous data associated with the panel item.

SEE ALSO

Inside Macintosh: Macintosh Toolbox Essentials 6-121 for more information on `inItemType`.

PSGetPanelItem

```
ShellErr PSGetPanelItem ( PanelPtr        inPanelPtr,
                          short           inItemNumber,
                          PanelItemType   *outItemType,
                          Handle          *outItemHandle,
                          Rect            *outItemRect );
```

DESCRIPTION

Returns the item specific data associated with a panel item.

PARAMETERS

`inPanelPtr`	A pointer to the panel
`inItemNumber`	The panel item number of the item we want to retrieve information about
`outItemType`	On return, the type of the requisite item
`outItemHandle`	On return, the handle of the requisite item
`outItemRect`	On return, the encompassing rect of the requisite item

RETURNS

`noErr`	The required information was retrieved
`kPanelItemNotFoundInListErr`	The item was not found on the panel

PSNewStyledStringItem

```
TEHandle PSNewStyledStringItem ( Rect     *inItemRect,
                               short      inSTRListRsrcID,
                               short      inStringListItem,
                               short      inFontNum,
                               short      inFontStyle,
                               short      inFontSize );
```

DESCRIPTION

Creates a new monostyled text item which can then be added to the panel with a call to `PSSetPanelItem`. The contents of a monostyled text item are loaded from a 'STR#' resource.

PARAMETERS

`inItemRect`	The encompassing rect of the styled string item on the panel
`inSTRListRsrcID`	The resource ID of the 'STR#' which the string is to be read from
`inStringListItem`	The index into the 'STR#' of the string required
`inFontNum`	The number of the font to draw the text in
`inFontStyle`	The style of the font to draw the text in
inFontSize	The size of the font to draw the text in

RETURNS

On success, returns a handle to the newly created styled string item. On failure, returns NULL.

SPECIAL CONSIDERATIONS

• Assumes that `inStringListItem` is greater than 0

PSNewStyledTextItem

```
TEHandle PSNewStyledTextItem (      Rect  *inItemRect,
                                    short inTEXTRsrcID );
```

DESCRIPTION

Creates a new multi styled text item which can then be added to the panel with a call to PSSetPanelItem. The contents of a styled text item are loaded from a 'TEXT' and optional 'styl' resource.

PARAMETERS

`inItemRect`	The encompassing rect of the styled text item on the panel
`inTEXTRsrcID`	The resource ID of the 'TEXT' and 'styl' resources to use

RETURNS

On success, returns a handle to the newly created styled text item. On failure, returns NULL.

SPECIAL CONSIDERATIONS

• Assumes that the 'TEXT' and 'styl' resources have the same resource ID.

• Assumes that `inItemRect` is specified in local co-ordinates of the current grafport.

PSNewUserItem

```
PanelUserItemHandle PSNewUserItem ( UserItemProcPtr   inUserItem,
                                    Boolean           inTransparent );
```

DESCRIPTION

Creates a new user item which can then be added to the panel with a call to `PSSetPanelItem`. User items can be used to create the writer’s own item on a panel, as with normal dialogs. Upgrader also provides routines for setting user items to DocViewer and static text items.

PARAMETERS

`inUserItem`	A pointer to the user item draw procedure
`inTransparent`	TRUE if the panel background (color, 'PICT's etc.) is to show through the user item. FALSE if the item’s background is whatever the user draw procedure sets

RETURNS

On success, returns a handle to the newly created user item on the panel. On failure, returns NULL.

SPECIAL CONSIDERATIONS

• Assumes that `inUserItem` is a valid user item procedure.

Sample code for creating a custom drawing procedure for a user item (error-checking is omitted for clarity):

```
ShellErr            theErr = noErr;
Handle              userItemHandle;
Rect                userItemRect;
PanelItemType       userItemType;
UserItemUPP         drawUPP = NULL;

// --- use    DisposeRoutineDescriptor(drawUPP)   later to kill it

drawUPP = NewUserItemProc(DrawingProcedureName);
if (drawUPP != NULL) {
       userItemHandle = PSNewUserItem(drawUPP, true);
       if (userItemHandle != NULL) {
              (void) PSGetPanelItem(gMyPluginPanel, kUserItemNumber,
                                    &userItemType, &userItemHandle, &userItemRect);

              (void) PSSetPanelItem(gMyPluginPanel, kUserItemNumber,
                                    userItemType, userItemHandle, &userItemRect);
       }
}
```

Document Viewer Routines

PSNewDocViewerItem

```
DocInfoHandle PSNewDocViewerItem ( PanelPtr      inPanelPtr,
                                   Rect          *inItemRect,
                                   DocViewerType inDocViewerType,
                                   short         inTextRsrcID,
                                   short         inBasePICTRsrcID );
```

DESCRIPTION

Creates a new DocViewer item which can then be added to the panel with a call to `PSSetPanelItem`. The contents can be read from 'TEXT' / 'styl' resources or from a SimpleText file, based on the value of the `inDocViewerType` parameter.

PARAMETERS

`inPanelPtr`	A pointer to the panel which will contain the new DocViewer item
`inItemRect`	The encompassing rect of the DocViewer item
`inDocViewerType`	`kDocResType` if the contents of the DocViewer item are to be loaded from resources, or `kDocFileType` if they are to be loaded from a SimpleText formatted file
`inTextRsrcID`	The 'TEXT' resource ID if `kDocResType`, or the 'flrf' resource ID for the file containing the text, if `kDocFileType`
`inBasePICTRsrcID`	For `kDocResTypes` this is the base resource ID of the first 'PICT' embedded in the text. For `kDocFileTypes`, `inBasePICTRsrcID` is ignored. (It is set to 1000 internally, which is the SimpleText standard).

RETURNS

On success, returns a handle to the newly created DocViewer item on the panel. On failure, returns NULL.

SPECIAL CONSIDERATIONS

• See *Appendix A* information on embedding 'PICT's into a DocViewer item.

PSHandleDocScroll

```
ShellErr PSHandleDocScroll ( PanelPtr     inPanelPtr,
                             Point        inLocalPt,
                             short        inDocItem );
```

DESCRIPTION

Handles scrolling in the DocViewer object.

PARAMETERS

`inPanelPtr`	The panel which contains the DocViewer object
`inLocalPt`	The point, in local coordinates, where the mouse was clicked in the DocViewer item. (Normally taken from the mousedown

	event record)
`inDocItem`	The panel item number of the DocViewer object

RETURNS

`noErr`	The scrolling was handled correctly
`kPanelItemNotFoundInListErr`	The DocViewer item was not found on the panel

SPECIAL CONSIDERATIONS

• Auto scrolling is not supported.

PSSaveDoc

```
void PSSaveDoc ( PanelPtr      inPanelPtr,
            short             inDocItem );
```

DESCRIPTION

Saves a DocViewer item to disk as a read only SimpleText format file. The standard Save dialog box is displayed, in which the user can optionally enter a path and filename for the DocViewer item. Default filenames for the standard Save dialog box can be provided for by putting a 'STR#' resource in the plug-in’s resource fork, with an ID of 1000 and the following fields:

```
resource 'STR#' (1000) {
        {
                $"CA",        // non-breaking space, used as the PICT delimiters
                "",           // (unused but must be present)
                "untitled",   // default "Save As" field in standard Save dialog
        }
};
```

PARAMETERS

`inPanelPtr`	Pointer to the panel which contains the DocViewer item to save
`inDocItem`	Panel item number of the DocViewer item

SPECIAL CONSIDERATIONS

• If the Standard File Package is not available, this routine should not be called, especially when booting from floppy disks. See the description of `PSCheckEnvironment` for information on how to check for the availability of the Standard File Package.

PSPrintDoc

```
ShellErr PSPrintDoc ( PanelPtr      inPanelPtr,
                      short         inDocItem );
```

DESCRIPTION

Prints a DocViewer item. The standard Print Job Dialog box is displayed. If the option key is held down when this call is made then the page setup dialog will be displayed.

PARAMETERS

`inPanelPtr`	Pointer to the panel which contains the DocViewer item to print
`inDocItem`	Panel item number of the DocViewer item

RETURNS

`noErr`	Printing completed successfully
`kPanelItemNotFoundInListErr`	The DocViewer item was not found on the panel
`PrError`	Standard printing error
`MemError`	Standard memory error

Navigation Routines

Three “navigation” routines are available to plug-in writers: `PSGoToPreviousPlugin`, `PSGoToNextPlugin` and `PSQuitShell`. In the normal course of events (i.e. if the plug-in has setup the Go Back / Continue buttons actions the Shell will take care of going to the next or previous plug-in, as defined in the sequence list or history list, when the user clicks on either the Go Back or Continue buttons (or their equivalents) on a panel. However, if the plug-in writer wishes to override the normal plug-in sequence for any reason, these routines can be used.

PSGotoPreviousPlugin

```
ShellErr PSGoToPreviousPlugin( void );
```

DESCRIPTION

Go to the current plug-in’s previous plug-in as defined by the plug-in history list. This routine calls the plug-in termination routine, `TerminatePluginModule`. `PSGotoPreviousPlugin` is called by the Shell when the user clicks on the Go Back button, or its equivalent, if there is a permitted previous plug-in to go to.

RETURNS

`noErr`	Always returns `noErr`

The following is a rough guide to the flow of control when the user clicks Go Back (or some equivalent):

Plug-in: Go Back
    -> causes ->
Shell: `PSGotoPreviousPlugin()`
    -> causes ->
Plug-in: `TerminatePluginModule()`

-> causes ->
Shell: checks the history list and loads the previous plug-in (if permitted)

PSGotoNextPlugin

```
ShellErr PSGoToNextPlugin( ResourceID inStrRsrcID );
```

DESCRIPTION

Go to the current plug-in’s next plug-in as defined by the plug-in sequence list, or go to the plug-in specified by name. This routine can either be called explicitly by a plug-in, in which case the 'STR ' containing the plug-in name to branch to is passed, or by the Shell (when the user clicks on the Continue button, or its equivalent), in which the name of the plug-in is taken from the plug-in sequence list maintained by the Shell. `PSGoToNextPlugin` calls the plug-in termination routine, `TerminatePluginModule`.

PARAMETERS

`inStrRsrcID`	`kUseDefaultNextModuleName` if using the default next plug-in as defined by the sequence list, else the resource ID of the 'STR ' containing the name of the plug-in to branch to

RETURNS

`noErr`	Went to the next plug-in successfully
`kUnknownPluginNameErr`	Trying to go forward from the last plug-in (no more plug-ins in the sequence list)
`kNextPluginSameAsCurrentErr`	Trying to call the same plug-in as we’re currently in
`MemError`	Standard memory error

The following is a rough guide to the flow of control when the user clicks Continue (or some equivalent):

Plug-in: Go Forward
-> causes ->
Shell: `PSGotoNextPlugin( kUseDefaultNextModuleName )`
-> causes ->
Plug-in: `TerminatePluginModule()`
-> causes ->
Shell: checks the sequence list and loads the next plug-in (if permitted)

The following is a rough guide to the flow of control when the plug-in explicitly calls `PSGotoNextPlugin`:

Plug-in: `PSGotoNextPlugin( resIDofPluginNameStr )`
-> causes ->
Plug-in: `TerminatePluginModule()`
-> causes ->
Shell: loads the plug-in specified in the 'STR ' referenced in `PSGotoNextPlugin`

PSQuitShell

```
ShellErr PSQuitShell( Boolean inCanAllowUserToContinue );
```

DESCRIPTION

Instructs the Upgrader Shell to quit after returning from the current handler.

PARAMETERS

`inCanAllowUserToContinue`	If `inCanAllowUserToContinue` is TRUE, the Shell will check whether or not a restart is necessary as soon as the Upgrader quits (most likely because system software has been installed). The global data type 'rsrq' (boolean) can be set to flag whether a restart or a quit is needed. If a restart is necessary a two button dialog will be displayed, one button to allow the user to cancel the quit instruction and continue with the Upgrader and the other button to allow the user to restart. If `inCanAllowUserToContinue` is FALSE, the Shell will check whether a restart is necessary as soon as the Upgrader quits but in this case will not display a dialog. If a restart is necessary it will simply restart. Under most situations `inCanAllowUserToContinue` will be set to TRUE by the plug-ins. It should only be set to FALSE in cases where a serious error has occurred and it would be unsafe to allow the Upgrader to continue.

RETURNS

`noErr`	There were no problems and the user did not select Continue if `PSQuitShell` displayed the Continue / Restart alert
`kUserContinuingAfterRestartMsgNum`	`PSQuitShell` displayed the Continue / Restart alert and the user selected Continue.

SPECIAL CONSIDERATIONS

• Upgrader does not quit immediately when this instruction is called, instead it sets a global to indicate that Upgrader is to quit as soon as possible. When control is returned to the Upgrader’s event loop, the global will be checked and the current plug-in’s termination routine is called. Other internal clean-up routines are then called before Upgrader finally quits. It is safe to call this routine from anywhere in the plug-in including the termination routine. (If it is called from within a plug-in termination routine Upgrader will not call the termination routine again).

Help Window Routines

PSSetupHelpWindow

```
OSErr PSSetupHelpWindow ( DocLocationType inLocation,
                          short           inRsrcID,
                          short           inBasePICTRsrcID,
                          Str255          inHelpPanelTitleStr,
                          PanelPtr        *outHelpPanelPtr );
```

DESCRIPTION

Called by a plug-in to setup the help window associated with that plug-in. If this is the first time that the routine has been called, it will create the `PanelPtr` and return it in `outHelpPanelPtr`.

PARAMETERS

`inLocation`	Location of Help panel contents. This will be either `kReadFromResourceFile`, if the help panel DocViewer items contents are being read from a resource, or `kReadFromSimpleTextFile`, if the contents are being read from a SimpleText format file.
`inRsrcID`	'TEXT' / 'styl' resource ID if `inLocation` is `kReadFromResourceFile` or the 'flrf' resource ID if `inLocation` is `kReadFromSimpleTextFile`
`inBasePICTRsrcID`	The resource ID of the first 'PICT' (if any) referenced in the text, if `inLocation` is `kReadFromResourceFile`. If `inLocation` is `kReadFromSimpleTextFile`, this parameter is ignored (as it's always set to 1000 internally).
`inHelpPanelTitleStr`	A pascal string containing the help panel title. There is no default title supplied, so if this parameter is not explicitly setup before being passed, the panel title will be some junk
`outHelpPanelPtr`	On return, a pointer to the help panel, or NULL if it couldn't be created

RETURNS

`noErr`	The help panel was successfully setup
`kInternalErr`	If the global help `PanelPtr` is NULL and we are attempting to change its contents
`kPanelItemNotFoundInListErr`	Invalid item number
`kCantChangePanelItemToSpecifiedTypeErr`	The item couldn't be changed
`MemError`	Standard memory error
`ResError`	Standard resource error

SPECIAL CONSIDERATIONS

• The 'TEXT' and 'styl' resources used to specify the text contents of the help panel (in the event of `inLocation` is `kReadFromResourceFile`) must have the same resource ID.

PSDisplayHelpWindow

```
void PSDisplayHelpWindow ( PanelPtr inHelpPanelPtr );
```

DESCRIPTION

Called by a plug-in to display the help window, or if it is already there, to bring it to the foreground. Also checks to see if (a) a printer is available, (b) the standard file package and list package are available and enables or disables the print and / or save buttons appropriately. See the description of `PSCheckEnvironment` for more information about checking for print / save capabilities.

PARAMETERS

`inHelpPanelPtr`	A pointer to the help panel

SPECIAL CONSIDERATIONS

• Assumes that `inHelpPanelPtr` is an initialized, valid pointer to the current help panel.

PSCloseHelpWindow

```
void PSCloseHelpWindow ( PanelPtr inHelpPanelPtr );
```

DESCRIPTION

Closes the help window, if `inHelpPanelPtr` is not NULL. This only hides the window, as it might be needed by another plug-in. As it's only hidden, its location and size are retained and don't need to be saved-off for use by another plug-in (the help panel is redisplayed by a future plug-in in the same state and position it was in before it was hidden).

PARAMETERS

`inHelpPanelPtr`	A pointer to the help panel

SPECIAL CONSIDERATIONS

• This is called by `PSHandleHelpWindowEvent` in response to the user clicking the Close button (or some equivalent action) on the help panel, therefore it does not need to be called by a plug-in in the normal course of events.

• The plug-in does not need to dispose of the help panel, as the shell takes care of it.

PSHandleHelpWindowEvent

```
Boolean PSHandleHelpWindowEvent ( PanelPtr      inHelpPanelPtr,
                                  EventRecord          *inEvent );
```

DESCRIPTION

Called by a plug-in in its event handler after determining that the event occurred in the help window to handle help window events. This is also called in the plug-in's resume event to give the shell a turn at handling the resume event for the help panel.

PARAMETERS

`inHelpPanelPtr`	A pointer to the Help panel
`inEvent`	A pointer to the latest event's record

RETURNS

TRUE if the event was handled by this routine, FALSE otherwise.

SPECIAL CONSIDERATIONS

• Assumes that `inHelpPanelPtr` is an initialized, valid pointer to the current help panel.

• Deals with mousedown events and resume events for the help panel.

Global Data Routines

PSSetGlobalData

```
ShellErr PSSetGlobalData ( GlobalDataType inGlobalDataType,
                           GlobalDataPtr  inGlobalDataPtr,
                           Size           inDataSize );
```

DESCRIPTION

Writes the value of the specified Global Data to the global list. If this Global Data doesn't exist, then it first creates it before updating the Global Data.

PARAMETERS

`inGlobalDataType`	A four character constant identifying the Global Data
`inGlobalDataPtr`	A pointer to the data to set
`inDataSize`	The number of bytes of data to write for this global

RETURNS

`noErr`	The global was successfully stored
`kGlobalDataOutOfMemErr`	There wasn't enough memory to add a new global

SPECIAL CONSIDERATIONS

• If the Global Data item already exists, then the data is overwritten, so it is the responsibility of the plug-in to ensure that the new data is the same type and length as the original, or else there might be a crash.

PSGetGlobalData

```
ShellErr PSGetGlobalData ( GlobalDataType inGlobalDataType,
                           GlobalDataPtr  inGlobalDataPtr,
                           Size           inMaxDataSize,
                           Size           *outActualDataSize );
```

DESCRIPTION

Reads the value of the specified global from the global list.

PARAMETERS

`inGlobalDataType`	A four character constant identifying the global
`inGlobalDataPtr`	A pointer to a buffer to copy the data into
`inMaxDataSize`	The number of bytes of data to read for this global
`outActualDataSize`	On return, the actual size of the stored global

RETURNS

`noErr`	The global was retrieved successfully
`kUnknownGlobalDataErr`	The global was not found or doesn't exist

SPECIAL CONSIDERATIONS

• It is the responsibility of the plug-in to allocate sufficient memory for the buffer before data is copied into it.

PSErrorAlert

```
SInt16 PSErrorAlert ( SInt16          inErrNum,
                      Boolean         inIsStandardShellErr,
                      ConstStr255Param  inParam0,
                      ConstStr255Param  inParam1,
                      ConstStr255Param  inParam2,
                      ConstStr255Param  inParam3,
                      SInt16          inDefaultButton,
                      SInt16          inCancelButton );
```

DESCRIPTION

Displays the text associated with the `inErrNum` in the related dialog, which is resized if necessary.

PARAMETERS

`inErrNum`	The error number of the error which occurred
`inIsStandardShellErr`	If TRUE, the error text, dialogs etc., are contained in the Shell. If FALSE, they are contained in the plug-in's resource fork
`inParam0`	Pascal string to replace ^0 in the dialog (or an empty pascal string if none)
`inParam1`	Pascal string to replace ^1 in the dialog
`inParam2`	Pascal string to replace ^2 in the dialog
`inParam3`	Pascal string to replace ^3 in the dialog
`inDefaultButton`	The item number of the default button on the dialog
`inCancelButton`	The item number of the cancel button on the dialog

RETURNS

The item number of the item the user selected on the dialog. Constants are defined for these return values in the "UpgraderPlugins.h" file.

SPECIAL CONSIDERATIONS

• Depending on the 'DLOG', there may or may not be, some or any ^0, ^1, ^2, ^3 strings to fill, so `inParam0`, `inParam1`, `inParam2` and `inParam3` therefore may or may not be required, depending on the 'DLOG' used. These parameters should be set to an empty string if not needed.

• If `inDefaultButton` is 0 (zero) then there is no default button set on the dialog. Similarly, if or `inCancelButton` is 0, there is no cancel button set on the dialog.

PSAlert

```
SInt16 PSAlert ( short            inDLOGID,
             DocumentType      inWhichFileContainsDLOG,
             ConstStr255Param  inAlertText,
             ConstStr255Param  inParam0,
             ConstStr255Param  inParam1,
             ConstStr255Param  inParam2,
             ConstStr255Param  inParam3,
             SInt16            inDefaultButton,
             SInt16            inCancelButton );
```

DESCRIPTION

General purpose alert display routine which displays `inAlertText` in the dialog `inDLOGID`, which is resized if necessary to fit the text. The plug-in developer may also create his / her own alert dialogs.

PARAMETERS

`inDLOGID`	Resource ID of the 'DLOG' resource
`inWhichFileContainsDLOG`	`kUpgraderFile` if the 'DLOG' resource is in the Shell, `kClientDataFile` if the 'DLOG' is in the data file, or `kCurrentPluginResFile` if the 'DLOG' is in the plug-in's resource fork
`inAlertText`	Pascal string containing the text to be displayed on the dialog
`inParam0`	Pascal string to replace ^0 in the dialog (or an empty pascal string if none)
`inParam1`	Pascal string to replace ^1 in the dialog
`inParam2`	Pascal string to replace ^2 in the dialog
`inParam3`	Pascal string to replace ^3 in the dialog
`inDefaultButton`	Item number of the default button on the dialog
`inCancelButton`	Item number of the cancel button on the dialog

RETURNS

The item number of the item the user selected on the dialog. Constants are defined for these return values in the "UpgraderPlugins.h" file.

SPECIAL CONSIDERATIONS

• Depending on the 'DLOG', there may or may not be, some or any ^0, ^1, ^2, ^3 strings to fill, so `inParam0`, `inParam1`, `inParam2` and `inParam3` therefore may or may not be required, depending on the 'DLOG' used. These parameters should be set to an empty string if not needed.

• If `inDefaultButton` is 0 (zero) then there is no default button set on the dialog. Similarly, if or `inCancelButton` is 0, there is no cancel button set on the dialog.

• If the plug-in writer designs his / her own alert box, there must be five buttons on the panel, irrespective of the number of buttons (the maximum number of supported buttons is five) the writer wishes to use. Any buttons not used should be hidden.

Utility Routines

This section details helpful utility routines available to plug-in writers. The various utility routines for the Upgrader includes `PSCheckEnvironment`, to check for Print / Save options, `PSReadFontInfo`, to read font information from an 'finf' resource, two more, `PSAlert` and `PSErrorAlert` to put up and handle an alert box / error alert box and also `PSCollect` , to preload any resources listed in a 'RES#' resource.

PSCheckEnvironment

```
OSErr PSCheckEnvironment ( EnvironmentType *outEnvironment );
```

DESCRIPTION

Perform a check on the machine’s runtime environment to verify the availability of certain facilities. Those supported at the moment are checks for

• Printing	- checks to see if an active printer is selected or available
• Saving	- checks to see if the List Manager Package and Standard File Manager Package are available

These checks are handy later on if the plug-in writer needs to enable / disable a Print / Save button on a panel, for example the help panel has both a Print and a Save button. This routine should be called on a resume event to check for a printer becoming available. In the event of booting from a floppy disk, the standard file and list package may not be available, so it might not be possible to save items.

PARAMETERS

`outEnvironment`	On return, a pointer to an `EnvironmentType` indicating whether or not printing and / or saving is available

RETURNS

`noErr`	Always returns `noErr`

SPECIAL CONSIDERATIONS

• The masks for checking availability of printing and saving are as follows:

Printing Available	:	`kPrinterAvailableMask`
Saving Available	:	`kStandardFilePackageAvailableMask`

PSReadFontInfo

```
Boolean PSReadFontInfo ( short      inLocation,
                         short      inRsrcID,
                         short      inFontItem,
                         short      *outFontFace,
                         short      *outFontStyle,
                         short      *outFontSize );
```

DESCRIPTION

Read-in font information from a font information ('finf') resource and return it in `outFontFace`, `outFontStyle` and `outFontSize`.

PARAMETERS

`inLocation`	One of: `kFontInfoInClientDataFile`, if the resource is in the data file, `kFontInfoInPluginFile`, if the resource is in the plug-in, or `kFontInfoInShell`, if the resource is in the Upgrader application
`inRsrcID`	The resource ID of the 'finf' resource
`inFontItem`	The index into the 'finf' resource for the required font
`outFontFace`	On return, the loaded font number
`outFontStyle`	On return, the loaded font style (face)
`outFontSize`	On return, the loaded font size

RETURNS

TRUE if the font was read successfully, FALSE if there was a problem

SPECIAL CONSIDERATIONS

• `outFontFace`, `outFontStyle` and `outFontSize` are undefined if either the 'finf' resource or the index into the resource weren't found.

NOTE

A 'finf' resource has the following definition (taken from *Fonts.r*)

```
type 'finf' {
        integer = $$CountOf(Fonts);                // Number of fonts
        array Fonts {
                integer;                           // Font Number
                unsigned hex integer  plain;       // Font Style
                integer;                           // Font Size
        };
};
```

PSLaunchFile

```
Boolean PSLaunchFile ( SInt16               inAppFileRefRsrcID,
                       SInt16               inDocFileRefRsrcID,
                       AEDescList           *inOptionalOpenParams,
                       Boolean              inLaunchAppInFront,
                       ProcessSerialNumber  *outApplicationPSN );
```

DESCRIPTION

Launches the specified file or application. If launching a file, the application the file is to be launched with must also be specified.

PARAMETERS

`inAppFileRefRsrcID`	The 'flrf' resource (see section 3.3 for more details), containing the application to be launched
`inDocFileRefRsrcID`	The 'flrf' resource of the document to be launched, or 0 (zero) if only an application launch
`inOptionalOpenParams`	A list of optional parameters sent in the 'odoc' AppleEvent
`inLaunchAppInFront`	If `kLaunchAppInFront`, launch the application in front of all other applications, if `kLaunchAppInBack`, launch the application in the background

`outApplicationPSN`	On success, the process serial number of the launched application is returned. This can be used to communicate with the launched application, or terminate the launched application. On failure, this parameter is undefined.

RETURNS

TRUE if no errors occurred, else FALSE.

PSMakeFSSpecFromFileRefID

```
Boolean PSMakeFSSpecFromFileRefID ( SInt16      inFileRefID,
                                    Boolean     inShowErrorAlert,
                                    FSSpec      *outFoundFile );
```

DESCRIPTION

Returns an FSSpec to a file defined by a 'flrf' resource. This allows plug-in writeres to store references to files using the standard 'flrf' resource, and find them during runtime.

PARAMETERS

`inFileRefID`	The ID of a 'flrf' resource (see section 3.3 for more details) containing the file to be found
`inShowErrorAlert`	If true, displays an alert if the file cannot be found.
`outFoundFile`	On success, a FSSpec describing the location of the found file.

RETURNS

TRUE if no errors occurred, else FALSE.

PSCollect

```
void PSCollect ( SInt16 inResListRsrcID );
```

DESCRIPTION

Given a resource list resource, or 'RES#', this routine goes through the list of resources and loads any of the resources in the list which aren’t already in memory. It also makes them non-purgeable.

PARAMETERS

`inResListRsrcID`	The resource ID of the 'RES#' resource

Resources

These resources are contained in the plug-in file.

The Error Mapping Resource ('ners')

One of the services the Upgrader provides to plug-ins is the ability to display error messages in automatically resizeable dialog boxes. The routines which provide this functionality are `PSErrorAlert` and `PSAlert`. `PSErrorAlert` makes use of 'ners' resources, which are used to identify individual error numbers with the error string to be displayed and the error dialog to display it in.

```
type 'ners' {
       wide array {
       integer;             // Internal error number
       integer;             // Index in 'STR#' ID = 500 resource
       integer;             // Dialog to display; 'DLOG' Rsrc ID
   };
};
```

The first integer will be the error number for which you want to display a message. The second integer is the index of a 'STR#' resource ID that the string is stored. The third integer contains the 'DLOG' ID of the error dialog in which the error message is to be displayed.

The Plug-in Format Resource ('pfmt')

The plug-in file format resource defines the version of the plug-in file, so the Upgrader can determine if it can execute the plug-in file or not.

```
type 'pfmt' {
   byte   PluginMajorRevisionNumber;             // The major version number
   byte   PluginMinorRevisionNumber;             // The minor version number
}
```

Summary of the Upgrader API

Constants

Constant	**Value**	**Comment**
`PSErrorAlert` / `PSAlert` constants:		
Dialog IDs , needed by plug-ins to define their own private 'ners' resources:		
`kStopOrContinueDLOGID`	`506`	*note, center main*
`kQuitOrContinueDLOGID`	`507`	*note, center main*
`kNoteOKDLOGID`	`508`	*note, center main*
`kRestartOrContinueDLOGID`	`514`	*note, center main*
`kStopOKDLOGID`	`519`	*stop, center main*
`kSkipOrContinueDLOGID`	`520`	*caution, alert parent*
`kStopAndQuitDLOGID`	`526`	*OK, center main*
`kStopDLOGID`	`600`	*OK, center main*
`kEmptyString`	`(ConstStr255Param) "\p"`	
`kOKButtonIndex`	`1`	*Default/selected button*
`kContinueNotSkipBtnIndex`	`1`	*Skip or continue*
`kSkipNotContinueBtnIndex`	`2`	
`kQuitButtonIndex`	`1`	*Continue or Quit*
`kContinueNotQuitBtnIndex`	`2`	
`kRestartButtonIndex`	`1`	*Continue or Restart*
`kContinueNotRestartBtnIndex`	`2`	

`kYesButtonIndex`	`1`	*Yes or no*
`kNoButtonIndex`	`2`	
`kStandardShellError`	`TRUE`	*Look in the Shell for the error*
`kPluginError`	`FALSE`	*Look in the plug-in for the error*
`kUpgraderFile`	`0`	
`kClientDataFile`	`1`	
`kCurrentPluginResFile`	`2`	

<u>`PSQuitShell` constants:</u>

`kDontAllowUserToContinue`	`FALSE`
`kAllowUserToContinue`	`TRUE`

<u>`PSGotoNextPlugin` constant:</u>

`kUseDefaultNextModuleName`	`0`

<u>`PSNewCustomPanel` constant:</u>

`kGrowWindow`	`0x02`	*`inFlags` parameter*

<u>`PSReadFontInfo` constants:</u>

Built-in 'finf' resource IDs for the `fontInfo` field of `StyledStringDesc`:

`kUpgraderFonts`	`128`
`kLargeTextStyle`	`0`
`kMediumTextStyle`	`1`
`kSmallTextStyle`	`2`
`kAlertTextStyle`	`3`

The file from which the 'finf' resource is to be read:

`kFontInfoInClientDataFile`	`0`
`kFontInfoInPluginFile`	`1`
`kFontInfoInShell`	`2`
`kFontInfoInAnyFile`	`3`

<u>`PSLaunchFile` constants:</u>

`kLaunchAppInFront`	`TRUE`	*Launch the file or app frontmost*
`kLaunchAppInBack`	`FALSE`	*Launch the file or app in the background*

<u>`PSRegisterPluginHandler` constants:</u>

`kEventHandlerID`	`'ehID'`
`kTerminationHandlerID`	`'thID'`

<u>Format Resource constants:</u>

`kSequenceResourceType`	`'tsqc'`

`kClientDataFormatRsrcType`	`'cfmt'`
`kPluginFormatRsrcType`	`'pfmt'`

<u>Error List Resource constant:</u>

`kErrorListRsrcType`	`'ners'`

<u>`PSNewDocViewerItem` / `PSSetPanelItem` constants:</u>

`kDocResType`	`'DOCV'`	*'TEXT'/'styl'/'PICT's stored in resources*
`kDocFileType`	`'DOCF'`	*SimpleText file (optional 'styl'/ 'PICT' in resource fork)*

<u>`PSSetPanelItem` constants:</u>

`kControlType`	`'CNTL'`	
`kIconType`	`'ICON'`	
`kUserItemType`	`'USER'`	*User panel item for drawing custom items*
`kStyledTextType`	`'STXT'`	*'TEXT' and 'styl' resource pairs*
`kStyledStringType`	`'SSTR'`	*'STR#', index and 'finf'*
`kPICTType`	`'PICT'`	

<u>Global Data Identifiers:</u>

Selected volume's vRefNum set by Target Selection plug-in (SInt16):
```
kTargetDiskVolRefNumDataType  'trgt'
```

Flag to signal to Shell and plug-ins that a restart is required (Boolean):
```
kForceRestartOnQuitDataType   'rsrq'
```

<u>`PSCheckEnvironment` constants:</u>

```
kPrinterAvailableMask         0x00000001
kStandardFilePackageAvailableMask   0x00000002
```

<u>`PSSetupHelpPanel` constants:</u>

```
kReadFromResourceFile         TRUE
kReadFromSimpleTextFile       FALSE
```

<u>`PSSetPanelItemAction` constants:</u>

```
kContinueButtonMask           0x0001
kGoBackButtonMask             0x0002
kQuitButtonMask               0x0004
kDefaultButtonMask            0x0010
```

<u>`PSShowPanel` / `PSHidePanel` constants:</u>

Pass one of these if the current global panel is unknown (e.g. if the plug-in doesn't display a panel):
```
kGlobalPanel                  (void *) 0xFFFFFFFF
```

```
kHelpPanel                    (void *) 0xFFFFFFFE
```

Data Types

Typedefs:

```
typedef     OSType             GlobalDataType;
typedef     Ptr                GlobalDataPtr;
typedef     OSErr              ShellErr;
typedef     OSType             HandlerIDType;
typedef     SInt16             ResourceID;
typedef     WindowPtr          PanelPtr;
typedef     UInt8              DVFlags;
typedef     unsigned long      PanelItemType;
typedef     PanelItemType      DocViewerType;
typedef     short              DocumentType;
typedef     Byte               DocLocationType;
typedef     unsigned long      EnvironmentType;
typedef     UInt16             PanelActionType;
typedef     Handle             PanelUserItemHandle;
```

Plug-in-Defined Routines

```
void InitializePluginModule(  void        *inPSTable,
                              SInt32            inRefCon,
                              Boolean     inEnterAtBeginning );

Boolean HandleEventForPluginModule( EventRecord *inEvent );

void TerminatePluginModule( void );
```

Upgrader Plug-in Routines

```
void SetupPlugin( void *shellFunctions );

ShellErr PSRegisterHandler( HandlerIDType       inHandlerType,
                            UniversalProcPtr    inHandlerProcPtr );

ShellErr PSHandleDocScroll( PanelPtr            inPanelPtr,
                            Point               inLocalPt,
                            short               inDocItem );

void PSSaveDoc( PanelPtr      inPanelPtr,
                short         inDocItem );

ShellErr PSPrintDoc( PanelPtr inPanelPtr,
                     short    inDocItem );

ShellErr PSSetGlobalData( GlobalDataType  inGlobalDataType,
                          GlobalDataPtr   inGlobalDataPtr,
                          Size            inDataSize );
```

```
ShellErr PSGetGlobalData( GlobalDataType  inGlobalDataType,
                          GlobalDataPtr   inGlobalDataPtr,
                          Size            inMaxDataSize,
                          Size            *outActualDataSize );

ShellErr PSSetupNewPanel( SInt16    inPanelItemsRsrcID,
                         PanelPtr   *outPanelPtr );

ShellErr PSNewCustomPanel( short    inPanelItemsRsrcID,
                           DVFlags  inFlags,
                           PanelPtr *outPanelPtr );

ShellErr PSUpdatePanel( PanelPtr inPanelPtr );

void PSDisposePanel( PanelPtr inPanelPtr );

ShellErr PSShowPanel( PanelPtr inPanelPtr );

void PSHidePanel( PanelPtr inPanelPtr );

Boolean PSGetPanelItemHit( PanelPtr       inPanelPtr,
                           EventRecord    *inPanelEvent,
                           short          *outItemHit );

ShellErr PSSetPanelItem( PanelPtr         inPanelPtr,
                         short            inItemNumber,
                         PanelItemType    inItemType,
                         Handle           inItemHandle,
                         Rect             *inItemRect );

ShellErr PSGetPanelItem( PanelPtr         inPanelPtr,
                         short            inItemNumber,
                         PanelItemType    *outItemType,
                         Handle           *outItemHandle,
                         Rect             *outItemRect );

void PSSetPRefCon( PanelPtr   inPanelPtr,
                   long       inRefCon );

long PSGetPRefCon( PanelPtr inPanelPtr );

ShellErr PSSetPanelItemAction( PanelPtr         inPanelPtr,
                               short            inItemNumber,
                               PanelActionType  inActionType );

TEHandle PSNewStyledStringItem( Rect      *inItemRect,
                                short     inSTRListRsrcID,
                                short     inStringListItem,
                                short     inFontNum,
                                short     inFontStyle,
                                short     inFontSize );

TEHandle PSNewStyledTextItem( Rect  *inItemRect,
                              short inTEXTRsrcID );

PanelUserItemHandle PSNewUserItem( UserItemProcPtr    inUserItem,
                                   Boolean            inTransparent );
```

```
DocInfoHandle PSNewDocViewerItem( PanelPtr      inPanelPtr,
                                  Rect          *inItemRect,
                                  DocViewerType inDocViewerType,
                                  short         inTextRsrcID,
                                  short         inBasePICTRsrcID );

ShellErr PSActivatePanel( PanelPtr              inPanelPtr,
                         Boolean                inShouldActivate );

ShellErr PSGoToPreviousPlugin( void );

ShellErr PSGoToNextPlugin( ResourceID inStrRsrcID );

ShellErr PSQuitShell( Boolean inCanAllowUserToContinue );

OSErr PSSetupHelpWindow( DocLocationType  inLocation,
                         short            inRsrcID,
                         short            inBasePICTRsrcID,
                         Str255           inHelpPanelTitleStr,
                         PanelPtr         *outHelpPanelPtr );

void PSDisplayHelpWindow( PanelPtr inHelpPanelPtr );

void PSCloseHelpWindow( PanelPtr inHelpPanelPtr );

Boolean PSHandleHelpWindowEvent( PanelPtr       inHelpPanelPtr,
                                 EventRecord    *inEvent );

OSErr PSCheckEnvironment( EnvironmentType *outEnvironment );

Boolean PSReadFontInfo( short inLocation,
                        short inRsrcID,
                        short inFontItem,
                        short *outFontFace,
                        short *outFontStyle,
                        short *outFontSize );

Boolean PSLaunchFile( SInt16              inAppFileRefRsrcID,
                     SInt16               inDocFileRefRsrcID,
                     AEDescList           *inOptionalOpenParams,
                     Boolean              inLaunchAppInFront,
                     ProcessSerialNumber  *outApplicationPSN );

Boolean PSMakeFSSpecFromFileRefID ( SInt16      inFileRefID,
                                    Boolean     inShowErrorAlert,
                                    FSSpec      *outFoundFile );

SInt16 PSErrorAlert( SInt16               inErrNum,
                     Boolean              inIsStandardShellErr,
                     ConstStr255Param     inParam0,
                     ConstStr255Param     inParam1,
                     ConstStr255Param     inParam2,
                     ConstStr255Param     inParam3,
                     SInt16               inDefaultButton,
                     SInt16               inCancelButton );

SInt16 PSAlert( short               inDLOGID,
                DocumentType        inWhichFileContainsDLOG,
                ConstStr255Param    inAlertText,
                ConstStr255Param    inParam0,
                ConstStr255Param    inParam1,
                ConstStr255Param    inParam2,
                ConstStr255Param    inParam3,
                SInt16              inDefaultButton,
                SInt16              inCancelButton );

void PSCollect( SInt16 inResListRsrcID );
```

Result Codes

The following are the errors numbers and constants defined in the Shell:

Error Name	**Number**	**Description**
DocViewer Errors:		
`kCantCreateDocumentErr`	7000	Unable to create the DocViewer object.
`kInvalidDocRecordErr`	7001	Returned if the `inDocData` parameter to any of theDocViewer routines is NULL.
`kInvalidFileSpecErr`	7002	Returned from `PSNewDocViewerItem` if the Upgrader couldn't resolve the file path while attempting to read from a SimpleText file.
General Shell error numbers that may be returned to plug-ins:		
`kUnknownPlugInHandlerErr`	1001	Returned from `PSRegisterHandler` if an invalid `HandlerIDType` parameter was passed.
`kCouldNotFindResourceMsgNum`	1003	General resource warning for plug-in use, not returned by any Upgrader routine to plug-ins.
`kMemoryErrorMsgNum`	1004	General memory warning for plug-in use, not returned by any Upgrader routine to plug-ins.
Errors Returned by `PSQuitShell`:		
`kUserContinuingAfterRestartAlertMsgNum`	1030	User selected Continue from Continue / Restart displayed during a call to `PSQuitShell`.
Plug-in sequence errors:		
`kUnknownPluginNameErr`	1040	Returned from `PSGoToNextPlugin` if the plug-in specified by `inStrResID` was

		invalid.
kNextPluginSameAsCurrentErr	1041	Returned from PSGoToNextPlugin if the plug-in specified by inStrResID is the same as the plug-in that the call was made from.

<u>Global data manager errors:</u>

kUnknownGlobalDataErr	2000	Returned from PSGetGlobalData if the type specified by parameter inGlobalDataType could not be found.
kGlobalDataOutOfMemErr	2001	Returned from PSSetGlobalData if memory could not be allocated for the inGlobalDataPtr parameter.
kUnsupportedPrefsFormatErr	2010	Message used by plug-ins when they find the preference resource for the plug-in is of a unsupported format , not returned by any Upgrader routine to plug-ins.
kNoPrefsErr	2011	Message used by plug-ins when they can't find the preference resource for the plug-in , not returned by any Upgrader routine to plug-ins.

<u>Panel manager errors:</u>

kPanelItemNotFoundInListErr	2050	Message indicating that an item could not be located in the panels item list, returned from PSHandleDocScroll, PSPrintDoc, PSUpdatePanel, PSGetPanelItem and PSSetPanelItemAction.
kCannotLoadNeededResourceErr	2051	General purpose resource message that plug-ins may use, is returned from PSSetupNewPanel and PSNewCustomPanel if problems occur loading resource referenced from the panel's 'DITL ' list.
kNoDataAvailableForItemErr	2052	Error number indicating that PSGetPanelItem failed to find information for the inItemNumber parameter, no message is defined for this in the Upgrader.
kInternalErr	2053	General purpose error message that may be used by plug-ins, returned by PSSetupNewPanel and PSNewCustomPanel if problems occurred while changing the panel list.
kItemTypeMismatchErr	2055	Error number returned by

	`PSNewDocViewerItem` if the `inDocViewerType` parameter is not of type `kDocResType` or `kDocFileType`.
`kCantChangePanelItemToSpecifiedTypeErr`	`2058` Error number returned by `PSSetPanelItem` if the `inItemType` parameter is one that the Upgrader doesn't support, no message is defined for this in the Upgrader.

Writing ModifierTool Editors

About ModifierTool Editors

ModifierTool provides the ability to modify Upgrader-based programs, such as the Install Mac OS 8 program used to install Mac OS 8. ModifierTool is designed to edit an existing Upgrader data file or create a new data file from scratch.

An editor is simply a PPC code fragment that is executed to present windows to edit the resources contained in the data file that are owned by the plug-in. The editor can be written using MPW, Metrowerks or other development environment, but the examples and utility files provided on the SDK use Metrowerks PowerPlant framework to make editor creation fast and easy. We encourage developers to use the file reference and text editors provided on the SDK so users can edit these common data types in a consistent way.

Writing a ModifierTool Editor

The best way to start an editor for a new plug-in is to duplicate an editor project for a similar plug-in. You'll find that most editors have the same basic resource reading / writing and display item setting and getting routines.

Editor Entry Point

When the editor is loaded and executed, the ModifierTool application turns over complete control to the editor until it it is finished. The editor should show modal or movable modal windows only, because if the user is allowed to switch back to the ModifierTool window, the main window will not handle the user interaction.

Listing 3-12 shows the entry point of the editor. You'll find this definition and other helpful routines in the "EditorUtilities.h" and "EditorUtilities.cp" files.

Listing 3-12 Editor parameter block definition

```
struct EditorLibProcParamBlock
{
   SInt16        fFormatNum;
   QDGlobals*    fQDGlobals;
   SInt16        fFileRefNum;
   SInt16        fPreferenceRsrcID;
   SInt16        fResListRsrcID;
};
typedef struct EditorLibProcParamBlock *EditorLibProcParamBlockPtr;

extern "C"{ typedef SInt32 (*EditorLibProcPtr)( EditorLibProcParamBlockPtr
inEditorLibProcParamBlockPtr ); }
```

Field descriptions

fFormatNum	The format of the EditorLibProcParamBlock structure. The only format currently defined is format 1.
fQDGlobals	A pointer to the ModifierTool application’s QuickDraw globals.
fFileRefNum	The file refnum of the data file resource fork. Upon launch of the editor the current resource file is set to the editor’s resource fork.
fPreferenceRsrcID	The lo-word of the RefCon value stored in the plug-in entry in the sequence resource. If this value is 0, then use the hard coded preference resource ID used by the plug-in.
fResListRsrcID	The ID of the 'RES#' resource that will contain the list of resources stored in the data file owned by the plug-in. The editor should update this resource whenever the user saves changes.

Upon entry, the editor should locate and read its preference resource then display a movable modal editing window. When the editing is finished, the editor should return one of four possible results:

0	The changes were saved and everything is great.
1	The user canceled the editing session, and any changes were discarded.
2	The user wishes to remove this plug-in. On return to the ModifierTool, the plug-in entry will be deleted.
-1 through -32768	An internal error occurred. Your editor should display an alert telling the user what the problem was before returning this result.

Updating Plug-in Resources

When the user wishes to save the changes he or she has made to the plug-in, the user should click Save in your editor window, which will cause your preference resource and the appropriate referenced resources to be saved. If the plug-in entry was just created by the user, and you need to create new referenced resources, then make sure to create this resources with IDs between 10,000 and 20,000. The routine `GetUniqueIDForResType` supplied in “EditorUtilities.cp” can help you generate these new IDs.

WARNING

Resources stored in the data file should never be shared between plug-ins. All referenced resources should have IDs between 10,000 and 20,000.

Two additional routines supplied in “EditorUtilities.cp” can help you update 'STR ' and 'STR#' resources. Use `WriteStringResource` to update a 'STR ' resource and `WriteStringListResourceIndex` to update an individual string index within a 'STR#' resource. The WriteStringListResourceIndex is not designed for speed, so if you need to update large 'STR#' resources, you may want to consider rewriting this routine.

When called, your editor is passed the ID of the preload list resource which you should update when saving changes. This resource is used to support running from a multiple disk set, such as floppies. The routines `ResetResListResource` and `AppendToResListResource` are provided in “EditorUtilities.cp” to help with this updating task. To use these routines, call `ResetResListResource` once to reset the list to zero entries, then call `AppendToResListResource` for each resource the your plug-in owns in the data file.

Removing Plug-in Resources

Because of the modular nature of ModifierTool and its editors, removing a plug-in entry is a joint effort between the editor and the ModifierTool application. When the user clicks Remove in your editor window, you should delete the preference resource and all referenced resources. The ModifierTool will remove the preload resource for you when it deletes the plug-in entry from the Upgrader-owned sequence resource.

The routine `DeleteResource` is provided in “EditorUtilities.cp” to help with deleting your resources.

Appendix A: Adding Pictures to SimpleText Documents

The following is an excerpt from *Technical Note: 1005: The Compleat Guide to SimpleText*.

So how do you use SimpleText to create Release notes? It’s easy. Get those creative juices flowing, grab a cup of strong coffee (or your favorite highly-caffeinated beverage), and read on.

Write the Text

You can handle this part yourself. Use any word processor or text editor that supports saving to text-only files (i.e., those files of type 'TEXT'). You can even use SimpleText if you so desire. Don’t put carriage returns after each line either, since SimpleText automatically wraps lines, just like a real word processor (the SimpleText window conforms to the size of the current screen, so don’t depend on the breaks you see either). Don’t worry about non-breaking space characters at this point either; you’ll get a chance to add them later. Just think about what pictures you want (if you want them at all) and in what order you want them. When you are finished with the text, save it as a text-only file. If your word processor gives you the option of putting carriage returns after lines or after paragraphs, choose the after paragraphs option.

SimpleText now lets you use different fonts, sizes and styles in your documents. No longer are you held captive to only one font. Be brave, spice up your document, this is a Mac, not a VT100. Just remember that people actually have to read this document, so don’t make it so cluttered with fonts and sizes that it’s illegible. Also stick to the standard fonts like Times, Helvetica, and Geneva, since if the font is not installed on the reader’s system, the text will end up in Geneva.

Draw the Pictures

First make a backup of your Scrapbook file (you should find it in your System Folder) if it contains anything you consider important. After backing it up, throw away the original copy (this makes things much easier later on in the process), but don't worry, if you made a backup you can use it to restore the original when finished. If you prefer, you can just rename the Scrapbook file, which effectively makes a backup copy.

Unfortunately, the ideal method for creating a picture involves both a paint program and a draw program. Once you are finished with your pictures, save them to a document, then do one of the following:

If you used a painting program to draw your pictures

1. Select your picture with a Lasso tool to ensure that only the minimum size of the image is copied. This takes up less space on disk and centers the picture in the document.
2. Copy the picture then paste it into the Scrapbook.
3. Repeat these steps for each individual picture you wish to include in the document.

If you used a draw program to draw your pictures

1. Copy each of your pictures into the Scrapbook.
2. Launch a paint program, then copy each picture from the Scrapbook into the paint program.
3. Once every picture is in a paint document, open the Scrapbook and clear each of your pictures from the Scrapbook. The Scrapbook should say "Empty Scrapbook" when you are finished (unless you did not start with a fresh Scrapbook).
4. Follow the procedure in the steps for a painting program to copy and paste each of your pictures back into the Scrapbook. At this point, regardless of which program you originally used to create your pictures, they should all be in the Scrapbook and in bitmap form (after being copied with a Lasso tool from a paint program). Because of a quirk in the Printing Manager and PostScript(R), you have to perform a few more steps.
5. Launch a draw program, then copy each picture from the Scrapbook into the draw program.
6. Once every picture is in a draw document, open the Scrapbook and clear each of your pictures from the Scrapbook. The Scrapbook should say "Empty Scrapbook" when you are finished (unless you did not start with a fresh Scrapbook).
7. Copy each picture back to the Scrapbook. This process makes the pictures "transparent" when printed, and this is important to avoid a problem with white, horizontal stripes running through your pictures.

Adding the Pictures

Launch ResEdit and open the text-only SimpleText document (you may want to work on a backup copy). SimpleText saves every document with a resource fork that holds the font information, so ResEdit should not warn you about the file not having a resource fork unless you created the document with a program other than SimpleText.

Open your Scrapbook file (the one with all the pictures in it). Its ResEdit window should contain a 'PICT' resource along with some others. Select 'PICT' (don't double-click), and copy this resource to the SimpleText document by bringing its window to the front and selecting Paste from the Edit menu. If you do this step correctly, your pictures and text should all be in the same document. Save the SimpleText document so you don't have to do this step again and close the Scrapbook.

Now you need to put the pictures into the proper numerical order so they show up in the correct order in the SimpleText document. Numbering starts at 1000 (i.e., first picture should be 1000, second picture 1001, etc.). To order these pictures, double-click on the 'PICT' in the SimpleText document's window. You should get another window which contains each of the pictures you copied into this document. Use the scroll bar until you find the first picture you want to appear in the document. Select it (by clicking on it once), and choose the Get Info or Get Resource Info option to get information on the resource. ResEdit displays an information window about the selected resource with space to enter a name and an ID (there is already a random ID number assigned). Change the ID to 1000 and give the picture a name too (i.e., "Figure 1", etc.). Near the bottom of this window you can see the resource attributes. Be sure that the "Purgeable" attribute is checked, then close the window. Repeat this process for each succeeding picture, giving each a successive number (i.e., 1001, 1002, 1003, etc.). When you are finished with all of the pictures, save the file and quit ResEdit.

That's the difficult part; the rest is icing. Go get some more coffee or whatever it is you are drinking.

Edit the Text to Make It Look Pretty With the Pictures

Launch SimpleText and open your document. Find the location where you want to place the first picture and position the text cursor there. Enter a carriage return or two (more if you want more space before the picture) then a non-breaking space character (Option-Space Bar, remember), which will be invisible.

Now resize the window, and voilà, when the window redraws, your picture will be just below the non-breaking space character. Now enter as many carriage returns as necessary to provide space for the picture. When you enter the first carriage return, SimpleText will erase the picture, so you will need to resize the window again to verify your spacing, clicking the zoom box works well.

Once you have enough room for the first picture (you probably want to leave an extra blank line or two after it too), move on to the next desired picture location and repeat the process. Continue this process (and don't forget to save the document along the way) until you have placed all of the pictures. When you finish placing the pictures, you should save the document again and try printing it on both an ImageWriter and LaserWriter if possible. You may wish to tweak the picture spacing or location to keep them from crossing printed-page boundaries.

When you are satisfied with the results, Quit SimpleText.

Making the File Read-Only

Make a copy of the file (to save a step if you decide to edit it again) then launch ResEdit. Now choose Get Info from the File menu and change the file type from 'TEXT' to 'ttro' (the lowercase is significant) and check to make sure the creator type is 'ttxt'. Now quit ResEdit and save the changes to the document when prompted.

That's all there is to it.

A Few Hints On Creating Good Documents With Pictures

The following hints should help to make your SimpleText document creation faster and more efficient as well as make the final document as nice as possible for the user.

- Always use the Lasso tool in paint programs to select pictures to appear in SimpleText documents; it makes them smaller.
- Keep pictures as small and simple as possible; the document takes up less room on disk and scrolling is faster.

- If two pictures appear on top of each other, you probably have two non-breaking space characters on the same line. Simply delete one to fix it. It is generally a good idea to put non-breaking space characters on a line by themselves with a blank line before it. In addition, always leave room for an extra line after the picture so you do not have the picture running into the text which follows it.
- If you need to use the non-breaking space character as a non-breaking space, you can.
- Since SimpleText assigns the numbered 'PICT' resources to the non-breaking space characters in the document, you can simply skip a resource number to use a non-breaking space character as a non-breaking space in the text. For example, if you had four non-breaking spaces in the document and you wanted pictures at the first, second, and fourth, you would number your 'PICT' resources 1000, 1001, and 1003. The third non-breaking space character would normally have 'PICT' resource 1002 assigned to it, but since there is not a resource with this ID, it simply acts as a non-breaking space in the document.
- Don’t worry about how horrible everything looks when you are editing; users will not be able to edit your document (unless they have read this Note), so they will not see the awful flashing, disappearing pictures, etc.
- Make the document read-only even if you do not use pictures. Distributing read-only documents to users gives the consistent impression that Release Notes are not to be modified.
- If your pictures are not appearing as you think they should, and if you cannot figure out what might be wrong by following the sequence in this Note, then try the following: Open the document with ResEdit. Click once on the 'PICT' list and choose Open Picker by ID from the Resource menu of ResEdit 2.x. You should get a window with a list of all of your pictures, in order, and numbered sequentially from 1000. If this is not what you get, then you have missed a step along the way and need to make sure all your pictures are in the resource and numbered sequentially.