FOUNDATION™

Version 1.0

LUNAR PRODUCTIONS

LUNAR PRODUCTIONS

Subject to the below stated limitations, Lunar Productions hereby warrants that the program contained in this unit will run on an Apple IIGS equipped as stated in the Requirements section of this document. Lunar Productions bears no responsibility for incompatibilities with non-Apple brand products, both hardware and software. Lunar Productions makes no warranties regarding merchantability or fitness for a particular purpose. Lunar Productions further warrants the media the software is provided on is free from error. To obtain this warranty, the enclosed purchaser registration form must be completed and returned to Lunar Productions within ten (10) days of purchase. All warranties are limited in duration to ninety (90) days from the date of the original retail purchase of this product. In the event of a faulty program or disk, the purchaser must send the original disk to Lunar Productions for replacement.

The warranty and remedies set forth above are exclusive, and in lieu of all others, oral or written, express or implied. No Lunar Productions dealer, agent or employee is authorized to make any modification, extension or addition to this warranty.

Apple Computer, Inc. MAKES NO WARRANTIES, EITHER EXPRESSED OR IMPLIED, REGARDING THE ENCLOSED COMPUTER SOFTWARE PACKAGE, ITS MERCHANTABILITY OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation may not apply to you. This warranty gives you specific legal rights, and you may have other rights which may vary from state to state.

FOUNDATION

This document was written, edited and composed on a desktop publishing system using Apple IIGS and Macintosh computers. The original text was composed using Claris Appleworks® 3.0. Screen shot illustrations were taken with NiftyList 3.4 by Dave Lyons and SavePic by Mark Collins. They were imported to Macintosh graphics using Imagery by Jeff Lewis and retouched for publication using Studio/8, published by Electronic Arts. Final editing and page layout were done using Aldus PageMaker 4.2. Proof and final pages were printed on an Apple Personal Laserwriter NT.

The Foundation Manual was written by
Mark Collins, Marc Wolfgram and Tammara Wolfgram
First Printing - Version 1.0 July 1992.

Foundation was conceived and written by
Marc Wolfgram, Jim Murphy and Mark Collins.

Special thanks to:

Apple Developer Technical Support
The Apple IIGS System 6.0 Development Team

Tim Swihart
Dave Lyons
Matt Deatherage

and of course, our critical beta testers!

Table of Contents

PREFACE

Foundation™ is a resource editor for the Apple IIGS® computer. It is designed to provide maximum power and speed, yet it provides safeguards to protect existing resource files and prevent resource corruption. The main program, consisting of the Foundation shell, ScriptEdit™ and HexEdit™ modules, is in itself sufficient to allow you to create or modify any standard or custom resource type. With the addition of "Native" editors, Foundation provides a complete WYSIWYG environment for resource manipulation. The level of technical knowledge required for its use is largely dependent on which type of editor you are using. Native editors will allow you to work on a resource with minimal background. ScriptEdit primarily requires that you understand the different types of data stored in a resource (number types, strings, resource reference ID's). Manipulating raw resource data with HexEdit requires extensive knowledge of resources.

Foundation is NOT a source code generation tool, nor is it a 'point and click' program builder. It is 'just a resource editor.' But, as the old saying goes, it does one thing, and it does it very well. No other environment on any platform that we have seen provides features like our "Intelligent Copy/Paste", dynamic resource naming and resource dependency manipulation. Many other functions can only be duplicated using APW Rez.

And that's not all... Over a year went into the creation of Foundation. We're going to continue working on native editors as well as additional features and enhancements to the system itself. And the adventure continues...

Lunar Productions
Marc Wolfgram
Mark Collins
Jim Murphy
Tammara Wolfgram

Requirements

To run Foundation you must have an Apple IIGS, ROM 1 or 3, Apple IIGS System Software 6.0 or later, and sufficient memory. We recommend at least two megabytes of RAM, as do the System 6 release notes, though we have successfully tested Foundation with as little as 1.5 megabytes. Foundation requires at least two 3.5" 800k drives, though for better performance, we recommend either a Hard Drive or AppleShare network. The volume containing Foundation must have sufficient space available for the creation of interim workfiles at least as large as the resource files you intend to edit.

Conventions Used In This book

The following visual cues are used throughout this book to identify different types of information:

◆ ***Note*** Notes like this contain information that is interesting but not essential for understanding of the main text.

△ ***Important*** Notes like this contain information that is essential.

▲ ***Warning*** Warnings like this indicate potential problems and things to avoid. Ignoring these warnings could have serious repercussions on your system.

New terms *are described or defined in little blocks of text on the side like this.*

This manual uses `Courier` type to represent code fragments and the names of procedures, functions, file names and error codes.

Where To Get More Information

We strongly recommend a thorough review of the Resource Manager and resource-related sections in Volume 3 of the Toolbox Reference and the System 6.0 Reference. Apple technical books, published by Addison-Wesley, such as the Apple IIGS Toolbox Reference, are available at many commercial book stores, as well as through Resource Central. Other manuals, technical notes and other materials of interest to Apple II and Apple IIGS users and programmers are also available from Resource Central.

Resource Central
P.O. Box 11250
Overland Park, Kansas 66207
(913) 469-6502
Fax- (913) 469-6507

If you plan to develop hardware or software products for sale through retail channels, you can get valuable support from Apple Developer Programs. Write to them at the following address. (Be sure to mention your need for Apple II Product support.)

Apple Developer Programs
Apple Computer Inc.
20424 Mariani Avenue
Cupertino, California 95014-6299

If you would like more information on Foundation or any of our products, or are interested in developing Foundation Resource Editor Modules, feel free to contact us at:

Lunar Productions
1808 Michael Drive
Waukesha, Wisconsin 53186
Attn: Program Development
(414) 549-9261

We also provide support on several online services. You can reach us at any of the following addresses:

***America Online**: M Wolfgram, JiMurphy3*
***Applelink**: D3672 (Lunar Productions)*
***GEnie**: Mark.Collins, A2Pro.Jim*
***Internet**: mwolfgram@aol.com*

CHAPTER 1 - FOUNDATION OVERVIEW

Resources

What are resources? Very simply, a resource is a program object that is accessed through the resource manager. For a somewhat oversimplified model, think of a resource as you would a file in a directory. When you want to load or run a file, you use GS/OS through whatever file system you're using to load the data from disk for manipulation. Resources are very similar. Just as different files have different file types based on their use or content, resources are subdivided into specific types. When an application uses a resource, it calls the resource manager to load it for use by that application.

String *refers to a specific data type used to store sequences of characters, such as words, phrases, sentences, etc. In Pascal, strings are a dedicated type. In C, strings are actually arrays of the data type Char or character. Other data types are used to store primarily numeric values, such as words and longwords. There are different types of strings used in resources. Among them are "Pascal Strings" or pStrings which begin with a byte value telling how many of the following bytes make up the string; "C Strings" which have no count parameter up front but instead end in a null character (Hex 0); "Word Strings" which are similar to pStrings but have a word length number (two bytes) to store the strings size;*

Most resources for the Apple IIGS are data resources - that is, they contain things like strings, menus, windows or other data sets, much like the different data files we have: word processing documents, graphic files, etc. There are executable or code resources as well, such as XCmd resources, that function as program routines. It is the job of the resource manager to provide the resources as needed. Without resources, windows, strings , menus and other data are stored within the program code itself. Changing windows, colors, menus, strings, etc. requires a lot of time-consuming recompliation/reassembly. Using resources, these changes can be made easily without rebuilding the application, since the program deals with the data most likely to change within independently referenced resources. Also, things become more portable. An XCmd written for one Hypercard IIGS stack can easily be transferred to another intact without having to rewrite Hypercard IIGS itself every time you want to gain the XCmd's functionality. Similarly, if you have a favorite method of creating or styling a Window, the Window and its dependent resources can easily be copied into another program for its use.

Resource Dependencies

Just as applications can utilize resources, some resources utilize other resources. Figures 1-1 and 1-2 illustrate an extreme example. The resource structure for it starts at the uppermost or "parent" level with a window template. The window template refers to a window color table (notice the white lined on black title bar), a string containing the title contained in the title bar, and a control list, as well as its own internal data for size, placement, etc. The Control List refers to two controls - the edit line control and the pop-up menu

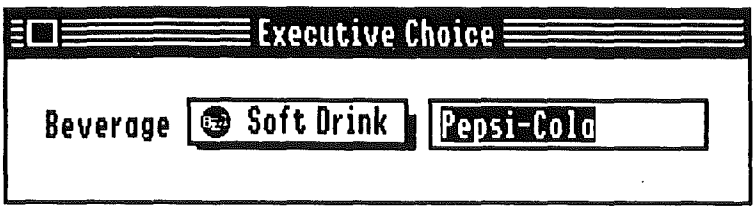


Figure 1-1 A Sample Window

Resource Dependency Tree

(for Sample Window Figure 1-1)

- `rWindParam1` *(Window Template)*
  - `rPString` *(the window's title string " Executive Choice ")*
  - `rWindColor` *(the window's Color Table)*
  - `rControlList` *(the window's Control List)*
    - `rControlTemplate` *(the LineEdit control)*
      - `rPString` *(the LineEdit's default string "Pepsi-Cola")*
    - `rControlTemplate` *(the pop-up control)*
      - `rMenu` *(defining the pop-up itself)*
        - `rPString` *(the menu's title string " Beverage ")*
        - `rMenuItem` *(the SoftDrink item)*
          - `rItemStruct` *(links the icon and the item string)*
            - `rIcon` *(the "Bottle Cap" icon)*
            - `rPString` *(the item string "Soft Drink")*
        - `rMenuItem` *(other items, each with more dependents)*

Figure 1-2 Diagram illustrating the Sample Window resource dependency structure

control. The edit line control references another string, while the pop-up menu control uses a Menu resource, which contains menu items, and... well, you get the picture. One big advantage to this system is that by changing one item, for instance the title string, the window itself can be changed without having to recompile the whole program. There is a potential disadvantage, however. By having other resources that a resource is dependent on, if one isn't as it should be, the rest of the structure is corrupted as well. If the title string resource is missing, the program would die an ignominious death due to a `resource not found` error. This somewhat hampers the portability of the resource. But, with Foundation's dependency handling system, there are safe-guards that can prevent this problem. If you want to delete a resource, the system first checks to make sure you aren't doing damage to something else. Similarly, you will be able to copy the window knowing that the rest of its baggage goes along with it.

Uses

What do you need a resource editor for? If you are a programmer, you could use it to create the resources your program needs to function well. With Foundation you no longer have to hard code your windows, leave your controls colorless, or keep your menus short so they fit into a Pascal string. Foundation lets you create your resources, and can also generate a listing for ready reference when you need to use the resources in your code.

If, on the other hand, you have no interest in programming, or simply haven't learned enough about it to feel comfortable with it, Foundation is still a very useful tool. You no longer have to be a programmer to have a program look the way you want. For example, if one of your favorite applications has a menu structure that you can't stand, using Foundation and the Window/Menu editor, you can shift the menus around to an order you can live with. Similarly, if someone gets a bit too creative in setting up all kinds of colored controls and windows, you can cure that problem too.

Editing Modes

Native Editor - A resource editor module that provide a WYSIWYG interface. These must be installed in the `Foundation.Edit` folder that is in the same directory as the main `Foundation` application.

There are three ways you can work on resources with Foundation, Native Editors, ScriptEdit or HexEdit. Native editors, when available, provide the greatest ease of use, because they are specifically crafted to edit the type of resource they work with. For instance, the Window editor will let you draw a window on the screen and place controls within it, just as if you were using a page layout program to size your page and place graphics and text.

ScriptEdit - A 'special' Foundation resource editor that lets you create and modify resources based on templates. Foundation comes with a supply of templates for many of the standard resource types, and you can create more with ***ScriptBuilder***.

ScriptEdit is almost as easy to work with as Native Editors. Though it does not let you maneuver items as objects, it allows you to work with resources broken down into their component parts - whether they are strings, point positions, rectangles, resource references, or whatever.

HexEdit - Another 'special' resource editor that allows you to directly manipulate the data contained in a resource on a byte level.

HexEdit lets you get 'down to the metal' with your resource. It is a Hex-ASCII type editor, allowing you to directly manipulate data on a byte by byte level. Just as a Hex-ASCII editor for disk drives isn't for everybody, we suggest caution when using HexEdit. It doesn't take much to corrupt a resource using HexEdit unless you know the format of the resource and you know precisely what you are doing.

Extensibility

As powerful as Foundation is already, you can easily expand it beyond the original package you have in your hands. Adding native editors is easy. Simply place the editor file (REM) in the `Foundation.Edit` folder and you're ready. The shell gets all the information it needs from the editor, even for resource types that haven't been defined when Foundation itself was released. There are currently several native editors available, and quite a few others being written. If you are a programmer, you might even want to try your hand at writing one. The Foundation Developer's Kit makes the job easier. If there is no native editor available for the resource type you want to work on, and you don't feel up to writing it yourself, you might try creating a script for the ScriptEdit module. The powerful editing tools in ScriptBuilder makes creating a script easy.

Launching Foundation

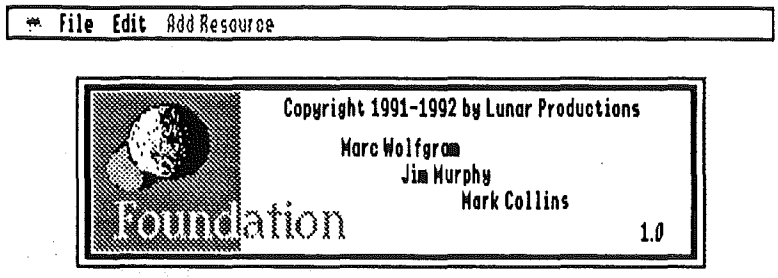


Figure 1-3 The Foundation Desktop with the Title Screen

Foundation is an AppleShare aware application that runs under any GS/OS compatible system launcher, including the Apple IIGS Finder, and the ORCA or GNO/ME shells. The `Foundation` application file and the folder `Foundation.Edit` must be located in the same directory. Foundation creates the directory `Foundation.User` where its private workfiles are stored. `Foundation.Edit` must contain the file `HexScriptREM`, which contains the HexEdit and ScriptEdit modules, as well as any other native editors you might have. You should make sure that the disk you launch Foundation from (or the User folder if launching from an AppleShare network) has sufficient room for a working copy of the resource fork(s) you plan to work on. When you successfully launch the program, the title screen will appear.

Working with Files

Once you've launched Foundation, you can either create a new file for your resources or open an existing file. Foundation will only allow you to open extended files - that is, those with a resource fork. Foundation 1.0 will not open or edit Macintosh format resource files (including the TrueType™ files used by Westcode's *Pointless*). Though these files also utilize a resource fork, and Macintosh style resources are used similarly, their resource structure is significantly different than the GS format.

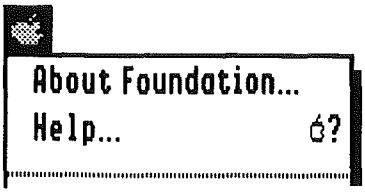


Figure 2-1 The ◆ Menu

The Shell - The main Foundation application. We call it the shell because it acts as an environment within which other tools work, such as HexEdit, ScriptEdit and Native Editors.

CHAPTER 2 - NAVIGATING FOUNDATION

The ◆ (Apple) Menu

About Foundation... displays the Foundation Credits Screen, showing the credits and version of the shell. If the current front window is an editor window, it displays this information on the editor instead. This dialog also provides a display of available memory.

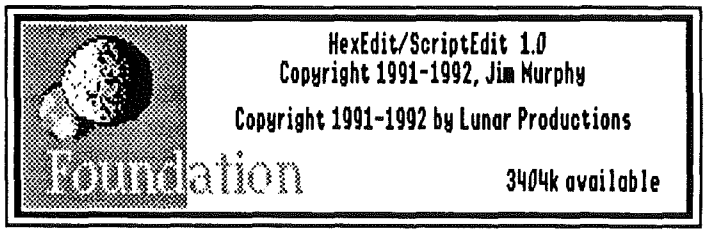


Figure 2-2 The Foundation About Window as seen when a ScriptEdit or HexEdit Window is Active

Help... brings up the Foundation help window. There are essentially two ways of getting help information. **Current Context,** the default, keeps track of what you are doing and

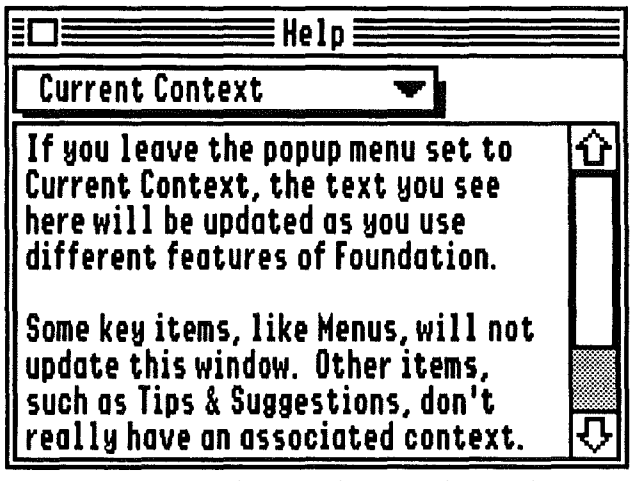


Figure 2-3 The Foundation Help Window

updates the help window content accordingly. For example, if you are using HexEdit, information on HexEdit will be displayed. The pop-up menu also allows you to get help with things not associated with a particular context, such as the menus, a handy reference of resource types, dependencies, and other generally helpful hints.

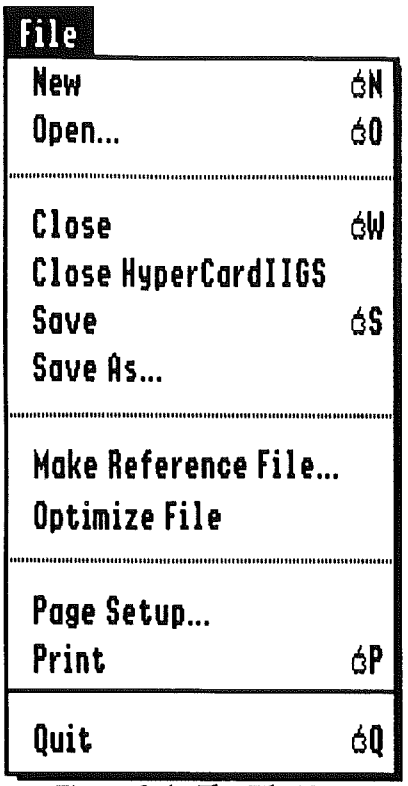


Figure 2-4 The File Menu

File Window *- Once a file is loaded or created for editing, the shell displays a File Window containing a list of resource types it contains. There may be up to four File Windows open at any given time, each one identified by a number in its title bar and by the color of the title bar itself. Items belonging to that file are also identified by the same number and color.*

Active File *- Foundation allows you to have up to four files open at any given time. You may only work on one file at a time. The Active File is the one being worked on or referenced at any given time. Upon opening or creating a file, that file is automatically the active file. To switch active files, all you need to do is bring one of the windows related to the file you want to make active (its File Window, one of its Type Windows or Selector Windows, or an editor working with an item from that file) to the front.*

The File Menu

New creates an empty working resource file for editing. You may have up to 4 files open at one time. These files will have the working title Untitled.a ... Untitled.d .

Open... loads an existing Apple IIGS resource file for editing. This file may be an application, CDev, HyperCard IIGS stack or any other type of file with an Apple IIGS formatted resource fork.

Close closes the currently active (front) window. If that window is a file window , you are asked to save any changes to the file, if any. Then, as it closes the file, it closes all windows associated with that file.

Close (FileName) closes the currently active file. (Its name is listed in the menu.)

Save writes the active file directly to disk, making all changes permanent. If you are working with a file created with **New** that has not been saved previously, selecting **Save** will operate identically with **Save As...**

Save as... first provides you with a Standard File dialog for naming/renaming your resource file, and then saves it just as in **Save**.

Make Reference File... creates a list of constants for referencing your resources by name in your program code. Only named resources are included in this list. The format of this file is selected through **Preferences...** in the Edit menu.

Optimize File recreates the current workfile, resource by resource. This has the effect of eliminating the holes created

Figure 2-5 Dialog Asking You To Confirm Optimizing A File.

by deleting and replacing resources, often providing a substantial savings in disk storage. You must select **Save** or **Save As...** to save the optimized file, as this command does not do it automatically. If the file contains resources that

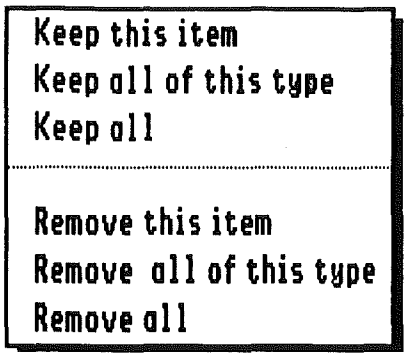


Figure 2-7 Popup Menu Listing The Options Available In The Optimize Option Window.

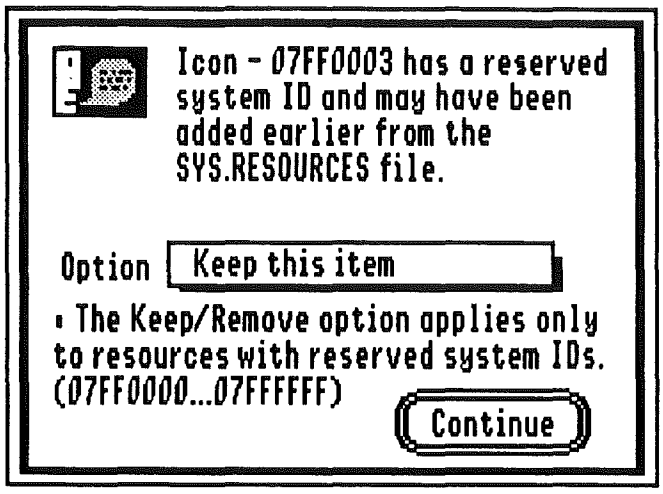


Figure 2-6 Option Window Presented When a Resource Numbered in the System Range Is Encountered by Optimize File

duplicate those contained in the system file `Sys.Resources`, Optimize lets you keep or remove them from the workfile.

Page Setup... and ***Print*** are provided only for editor use.

Quit is selected to end a Foundation session. If any files are open with changes that have not been saved to disk, you will be prompted to save them before the program quits.

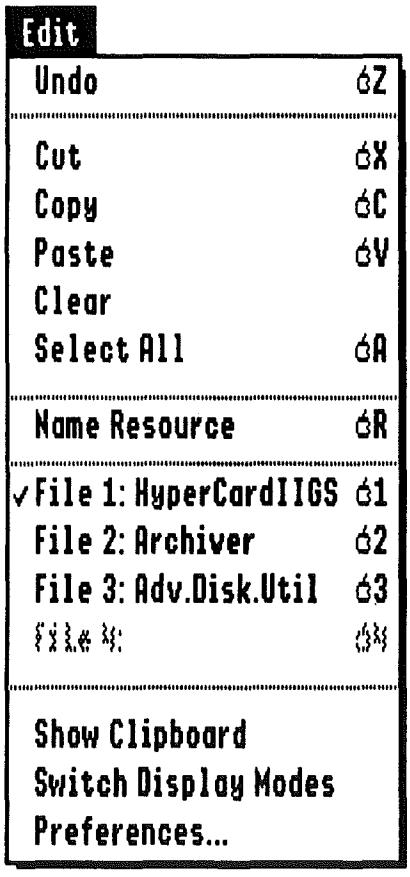


Figure 2-8 The Edit Menu

The Edit Menu

Undo, Cut and ***Clear*** are provided primarily for support of New Desk Accessories, though some native editors use them as well.

Copy and ***Paste*** are used universally throughout Foundation for copying and pasting items. From NDAs and editors, they are used to copy and paste items of the same basic type (text, pictures, sounds, etc.). A special feature of Foundation is the intelligent copy/paste function which lets you copy an item complete with all its dependent items. Copying a resource with its dependents is done by selecting **Copy** when a type window or native selector window is the front window and one of its items is selected. This copy does not look for 'Parent' resources of the selected item. If you want to copy the structure with the 'Parent' and its dependent items, you will need to perform the copy on the 'Parent' item instead. In order to paste, a file or type window must be the front window. When this paste is done, any resource number conflicts are resolved.

Select All is provided for editor use.

Name Resource brings up the 'Namer' window. This window is used to create, change or delete resource names.

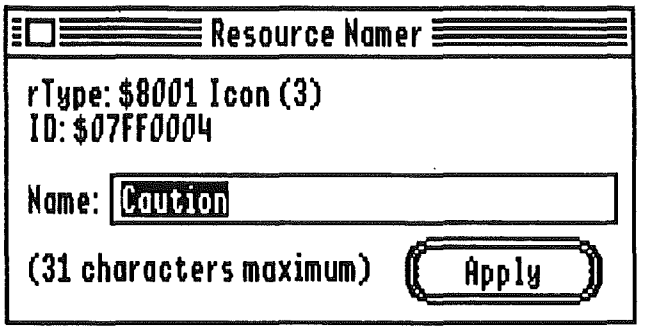


Figure 2-9 The 'Namer' Window

You can leave it on the desktop and the resource it refers to is updated based on the currently active item. Since Foundation maintains resource names internally, this window is the only way you can edit them.

File 1: ... ***File 4:*** brings the File window of the selected file to front, making it the active file. As you work with up to four files, their file names will appear in the menu. Unused file entries in the menu will otherwise be dimmed and inactive.

Show Clipboard displays the current contents of the clipboard. Text and picture scraps are displayed as text or

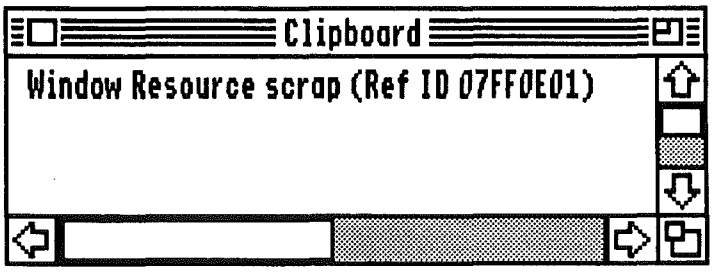


Figure 2-10 The Foundation Clipboard Window

pictures. Any other standard scraps are described. For example, Figure 2-10 shows what the clipboard looks like when the current scrap is a resource reference scrap.

Switch Display Modes toggles Foundation between 640 and 320 Super HiRes graphics modes. Note that some functions and editors (for example HexEdit and ScriptEdit) are not available in 320 mode. While the switch is taking place, the intervening text screen displays any actions it takes to adjust window position and availability as needed for the switch. Items available only under 640 mode are hidden and the remaining windows are adjusted proportionally as appropriate for the new screen mode.

Preferences... lets you set a number of program parameters. The first popup menu lets you choose whether the list of resource types displayed in a file window will be sorted alphabetically by the name of the resource type, or numerically. The next popup similarly determines the order of items displayed in a type window either numerically by the item ID numbers or alphabetically by their resource name.

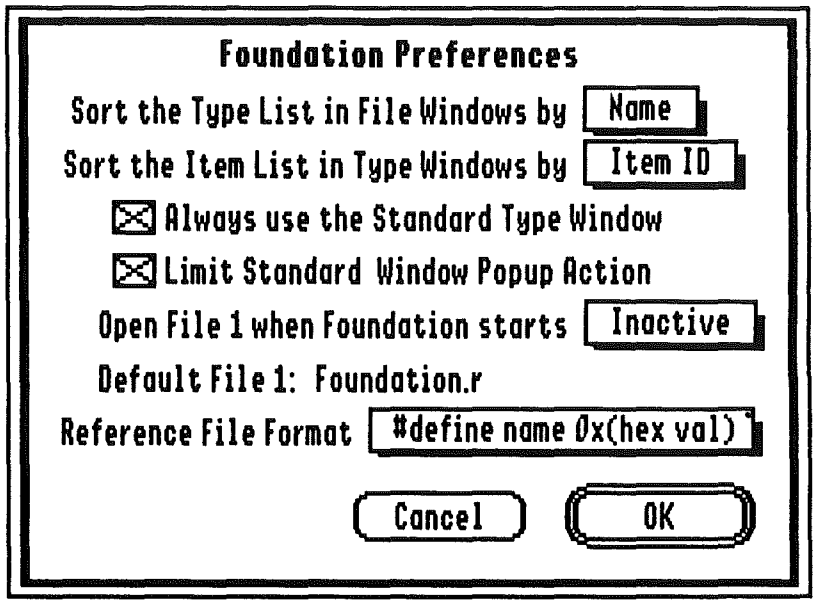


Figure 2-11 The Foundation Preferences Window

The first checkbox forces use of the standard type window, even if a native editor provides a native selector. To use native selectors, be sure this box is not checked.

The second checkbox determines the action of the popup menus in the file and type and item dependency windows. For example, if you have an item selected in a type list, not having this box checked will let you initiate an action simply by selecting a pop-up choice. If the box is checked, only double-clicking on the list item executes the current action.

Foundation can automatically open a file when starting up. By setting the "Open File 1..." popup to **Default**, the file listed as Default File 1 will be opened. If this is set to **Inactive**, no file is automatically opened. Selecting **Choose...** lets you pick the file to load automatically using a standard file dialog. If Foundation can not find the default file at startup, no file is loaded and normal operation continues.

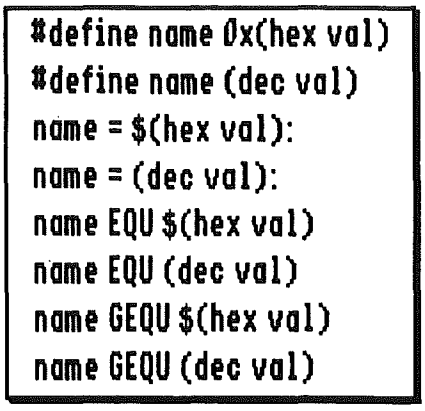


Figure 2-12 Pop-Up Menu Showing Different Formats for Reference File Output

The last popup menu lets you select the format of the reference file you can create using **Make Reference File...** These selections let you pick the format most appropriate for the language you are programming with. For example, if you are programming in ORCA/M assembler, you might select **name GEQU $(hex val)**, and then create a reference file that contained entries like this:

```
MyAppleMenu          GEQU $00000001
About_MItem          GEQU $00000001
Help_MItem           GEQU $00000002
```

Foundation substitutes underscore '_' for spaces in names.

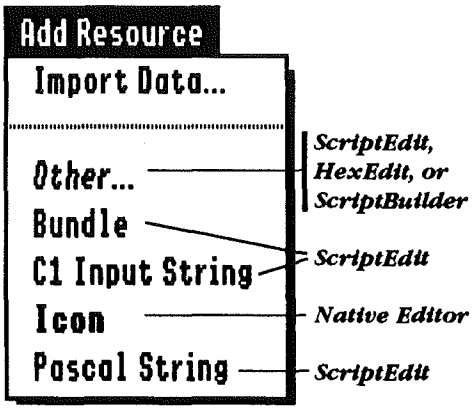


Figure 2-13 The Add Resource Menu

The Add Resource Menu

Just as there are different types of editors, there are several ways to add new resources as well.

Import Data... lets you import of the data fork of a file into the active file as a resource. After selecting the file to import you will be asked for a resource type and ID to use. If the ID is already in use, a unique ID is automatically assigned. Foundation performs no validation on the data loaded, so be sure to import data that is appropriate for the resource type.

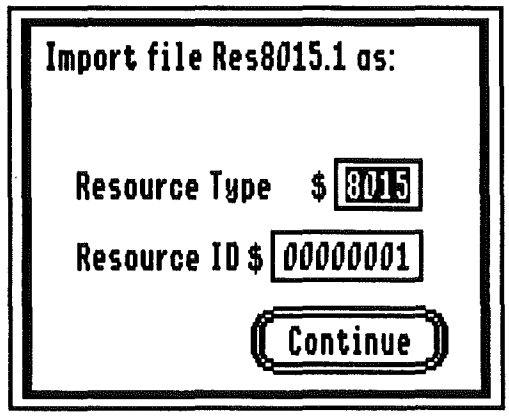


Figure 2-14 Import Item Type and ID Assignment Window

Other... lets you add a resource using HexEdit or ScriptEdit, or define a new ScriptEdit template. The different editor types are explained in Chapter 4.

The remaining entries in this menu are defined resource types. Selecting one of these types will create a new resource of that type and open an editor to work on the new item. A native editor is used for **bold** resource types, while a ScriptEdit template is used for the others.

CHAPTER 3 - WORKING WITH FOUNDATION

There are several windows that are key to using Foundation, some of which have already been discussed somewhat briefly. This chapter explains them in detail.

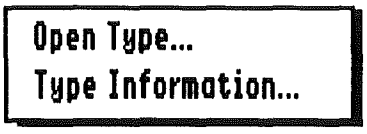


Figure 3-2
The File Window Action Menu

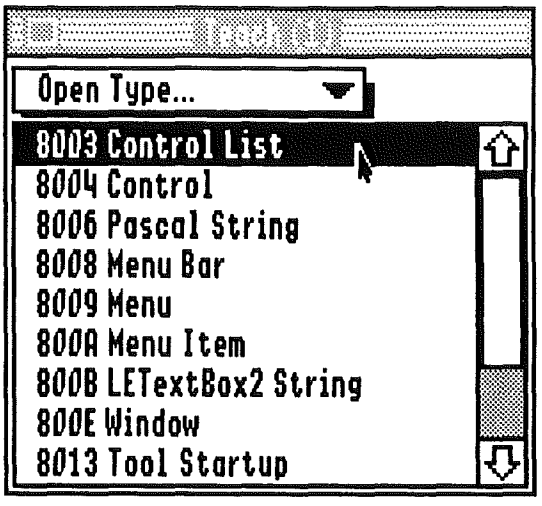


Figure 3-1 A File Window

The File Window

Any file open for editing (using **New** or **Open...**) will be represented on screen by a file window. Each file is identified by the number shown in parentheses next to the file name, and by the color of the window title bar. Any window associated with this file will similarly be identified by file number and color. Selecting any window associated with a file will make that file the active file. You may have up to four files open at a time.

The file window has two main elements: a pop-up action menu and a list of all the resource types currently contained in the file. Resource names are maintained internally by Foundation. For this reason, they can only be edited through the Namer window, and `8014 Resource Names` will never appear in the type list. To select a type to perform an operation on, simply double-click on it. Figure 3-2 illustrates the two choices in the action menu.

◆ ***Note*** **You can optionally execute the action on the selected type if the preference to limit pop-up action is NOT checked. See the section on Preferences ... in Chapter 2 for more details.**

Open Type... opens a type window listing all the items in the file of the selected resource type. Similar in appearance to the file window, the type window is the gateway to resource editing. Like the type window, it has an action pop-up and

a list. Items are listed by resource ID and resource name. Items without names will only appear by ID. Type window pop-up and list operation is the same as the file window.

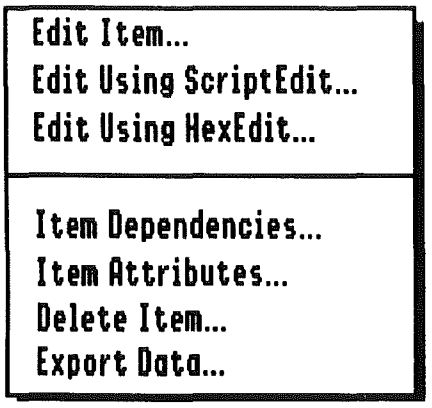


Figure 3-3
The Type Window Action Menu

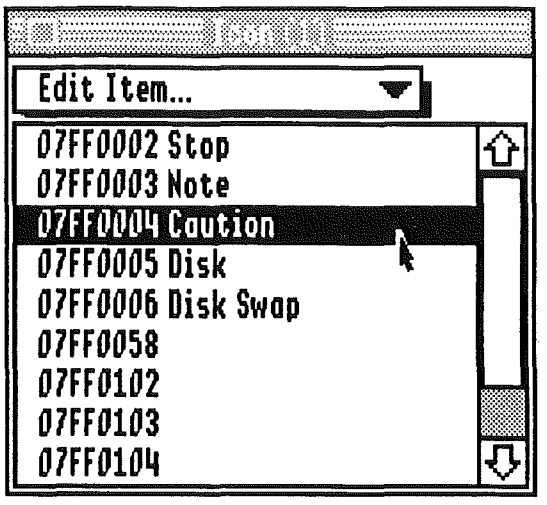


Figure 3-4 The Type Window

Type Information provides a summary of the selected type, includes a resource count, the total number of bytes these items consume, and their average size.

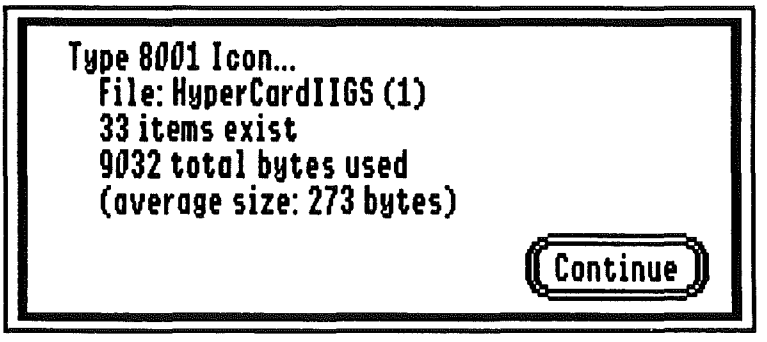


Figure 3-5 The Type Information Window

The Type and Dependency Window Action Menu

The action menu in the type window (Figure 3-3) controls the operation to be performed on the selected item.

Edit Item... calls a native editor. If a native editor is unavailable for the type, **Edit Item...** is dimmed.

Edit Using ScriptEdit... will initiate an editing session with a ScriptEdit template. You will be informed if no script exists for the selected item's type and can continue editing the item in HexEdit.

Edit with HexEdit... Since HexEdit can be used to edit any resource, selecting it will always begin a HexEdit session.

△ *Important* **HexEdit and ScriptEdit are not available in 320 SHR Mode. These selections will be dimmed in that mode.**

Item Dependencies... allows you to walk through an item's resource dependency tree. The dependencies window has four main items. At the top is the same action pop-up used in the type window, except that **Delete Item...** is disabled. Below the pop-up are three separate controls that work together to provide a single parent/item/dependent snapshot.

Dependency Tree - A resource hierarchy starting with a parent resource, which uses other resources (the parent's dependents), some of which may use other resources (the dependent's dependents), etc.

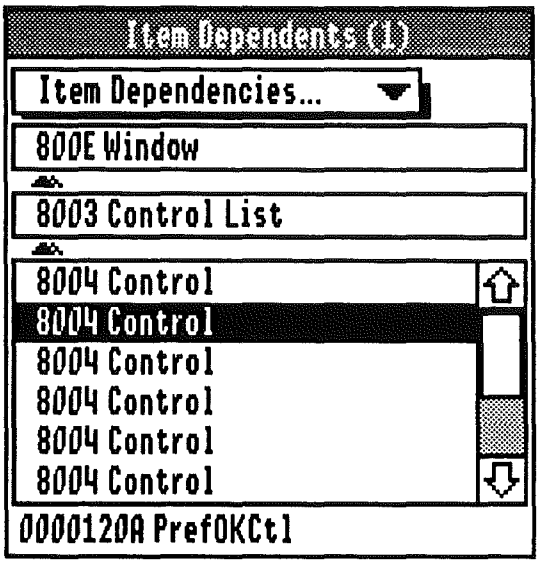


Figure 3-6 - The Dependency Window

First, the parent box displays the resource types of the item's immediate parent. Next, the item itself is shown it the item box. Last, a list of the item's immediate dependent resources is displayed. At the very bottom of the window the resource ID and name of the selected item is displayed. By selecting **Item Dependencies...** for the parent or one of the dependents lets you step all the way up and down a dependency tree. For example, if you originally selected **Item Dependencies...** on a Control List, the parent box would show the `800E Window` and the list would contain the `8004 Control` resources contained in the Control List. Double-clicking on the parent box would make the `800E Window` the current item, updating the Item Dependents window to display the Control List in the dependents list along with perhaps a `8006 Pascal String` title, and a `8010 Window Color Table`. Since no resource uses windows, the parent box will display `(No Parent)`. The same is true as you move down the tree. Selecting **Item Dependencies...** for a `8006 Pascal String` would display `(No Dependents)` in the list. See Appendix B for the various standard resource dependencies.

Item Attributes lets you change an item's resource ID and set the various attributes that determine how the Resource and Memory managers handle the item. Refer to the Apple IIGS Toolbox Reference, Volume 1, "Memory Manager", and Volume 3, "Resource Manager", for information on the function of these attributes.

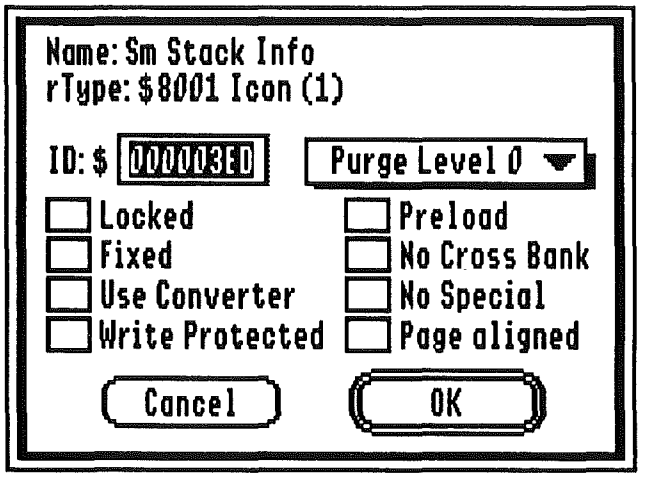


Figure 3-7 - The Attributes Window

▲ ***Warning*** **Changing the ID of an item can be extremely hazardous. In an existing application, such a change may cause a** `resNotFound` **error when the item is used, since its reference within the file is most often based on its ID. This problem also occurs within a dependent resource tree, since reference is always by ID. Foundation does no validity checking to ensure the dependency compatibility of an ID number change.**

Delete Item... lets you remove a resource from a file. Since deleting resources is something that should be done with extreme care, a confirmation dialog appears just to make sure that you really want to delete the selected item. If the item you are about to delete can possibly be part of a dependency

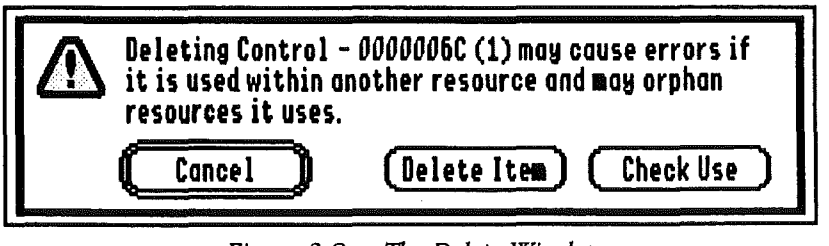


Figure 3-8 - The Delete Window

tree that dialog includes a **Check Use** button. If you choose Check Use, another window appears reporting the item's dependency status. If Foundation finds that the item is part of a dependency tree, the top-most parent is resolved and displayed. The delete options are limited to removing the

entire tree from the parent or removing the selected item with or without its dependents.

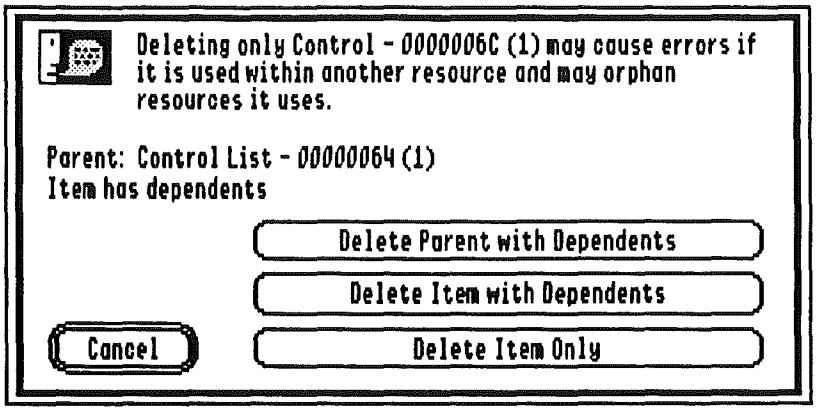


Figure 3-9 - The Check Uses Window

▲ ***Warning*** **Deleting an item from an existing application may lead to a system crash resulting from a** `resNotFound` **error when application attempts to use that item. Similar results will occur when deleting an item used by another resource, which is why we strongly encourage you to check the item's use before deleting. Resources may be used by more than one resource.**

Export Item... lets you save the data contained within a resource as a data fork file. This can be useful for translating text to a format editable in a word processor.

Three Editing Modes

As you have already discovered, there are three modes, or levels, of resource editing in Foundation. Native editors provide natural editing of resources. ScriptEdit represents resources as abstract data. HexEdit allows byte-level editing. Foundation version 1.0 is provided with ScriptEdit and HexEdit only. ScriptEdit, however, can provide 'native' editing capabilties for many standard resource types.

Native Editors provide a WYSIWYG resource editing environment. For example, an Icon native editor would let you draw the icon in a FatBits mode. As they become available, each native editor will have its own documentation.

◆ *Note* **Although Foundation version 1.0 does not come with any native editors, several are either available or under development at this time. For more information on native Foundation editors, contact Lunar Productions. If you have returned your registration form, you will be notified when updates and new editors are available.**

ScriptEdit is a general editor which can be used to work with any structured resource type (i.e. not a code resource). It uses special scripts, or templates, to define not only the item's structure but also the way it is presented for editing. Foundation comes with a selection of scripts already provided for most of the defined system resource types. ScriptBuilder lets you create your own variants of these scripts, and also lets you build templates for other types as well.

Figure 4-1
The ScriptEdit Menu

The ScriptEdit menu (Figure 4-1) is used to create or edit scripts.

New Script... creates a new, empty script window. To adjust the size of this window, move the cursor to the bottom or right borders of the window and the cursor becomes a grow box. While this grow box cursor is active, you can change the size of the window just as if it had a grow box in it. When the script is saved, its size and position are stored as part of the script.

New Card adds another card to a script. The card number ("xx of nn") indicates which card of a script you are on, and if there is more than one card in the script, two arrow buttons

will let you page through the cards. A single window can expand automatically to multiple ScriptEdit windows, depending on the data. For example, a script for an `rControlList` would be defined by a loop field and a resource ID field. By adding control references (the resource ID field), you can reach a point where the card becomes filled. ScriptEdit creates additional cards as needed to accomodate the data. There is no limit to the number of cards a script can contain.

Loop *is a hidden control field in a script that lets you define a repeating structure.*

Resource ID Field *is a special field that lets you either enter a resource ID by typing or selecting it from a list of resources that are currently available.*

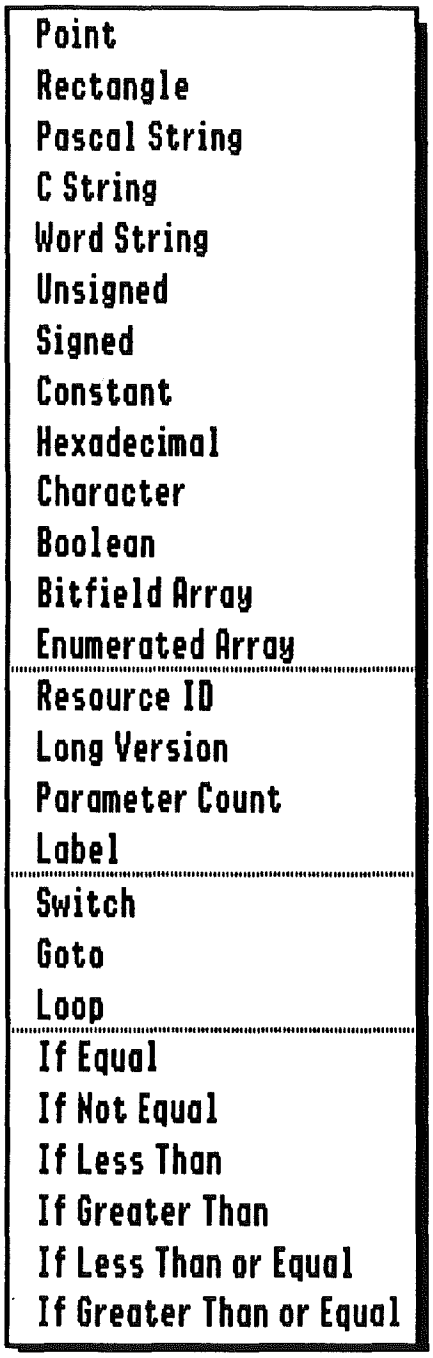


Figure 4-2 - The ScriptBuilder New Field Pop-Up Menu (See Appendix A for descriptions)

Delete Card removes the card currently displayed, along with any fields contained on it. If there is only one card remaining in the script, this will remove the fields on that card, leaving a empty card.

New Field... brings up the ScriptBuilder control window (Figure 4-3). Select the type of field you wish to add from the popup menu (Figure 4-2 shows this menu) and hit the **Add** button. Your new item will appear in the window. You can place this item anywhere in the window just as you can place file icons within the Finder. Double-click on the item to set the characteristics of the field . For example, if the field is a pascal string, you can choose whether it is type limited (a pascal string by definition has a maximum of 255 characters) or a fixed length string.

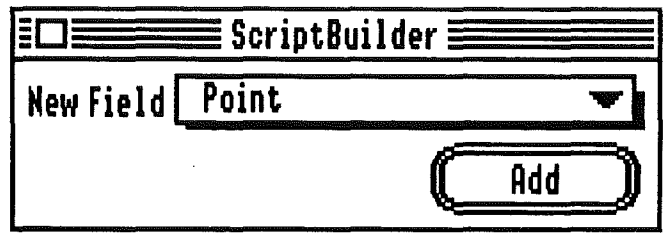


Figure 4-3 - The ScriptBuilder Control Window

Lightning Copy is a method of adding a new field without accessing the Script Builder control window. To use it, hold down the control key while clicking and dragging any field on the current card. This creates a clone of that field.

Link Tool selects the destination field for a looping or branching field. It is used to link one field with another. To use it, hold down the command key while clicking on a loop, goto or switch command field. The cursor will change to the link tool allowing you to select the destination field. To cancel a link, click somewhere in the window that is not a field. The link cursor will change back to the regular cursor, indicating that the link did not occur.

Delete Field lets you remove the currently selected field (indicated by a surrounding marquee rectangle) from a script.

Reorder Fields... lets you alter a script's parsing order (the order used by ScriptEdit when it is using this script). This order can be seen by holding down the command and option keys simultaneously when a script window is frontmost. A small filled and framed rectangle is drawn, centered, on top of each field. Within the rectangle is an integer indicating field order. This order is important since it is also the order ScriptEdit uses to match a resource's data to the script. If this order is incorrect, the data will be presented and written incorrectly, invalidating the resource structure.

Figure 4-4 illustrates one simple example of a ScriptEdit window showing what an `rTwoRects` template might look like. Appendix B illustrates that this resource structure is made up two rects, each of which is made up of four words

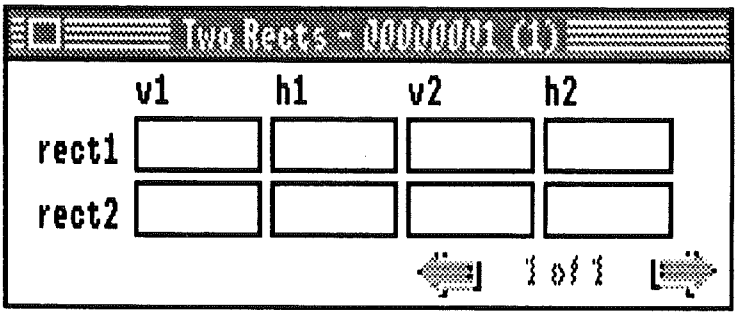


Figure 4-4 A Sample ScriptEdit Window

or integers. These integers represent the coordinates of the opposite corners of the rectangle. To build the `rTwoRects` script, you would need to add two **Rect** fields, and then add the six labels to show what data belongs in each entry rectangle. To use this template to create an `rTwoRects` resource, enter the horizontal and vertical coordinates in the entry rectangles provided. That's all there is to it. ScriptEdit makes it easy to create any other data resource type.

HexEdit offers the most basic and direct method of creating and manipulating resources. There are no guides or structures to help you ensure data integrity here. Just enter raw data.

The HexEdit window in Figure 4-5 displays two different views of the same resource data - hexidecimal bytes and ASCII characters. You can edit the resource using either side of the display one byte at a time. You can use the mouse to select a byte on either the hex or ASCII views. Alternatively,

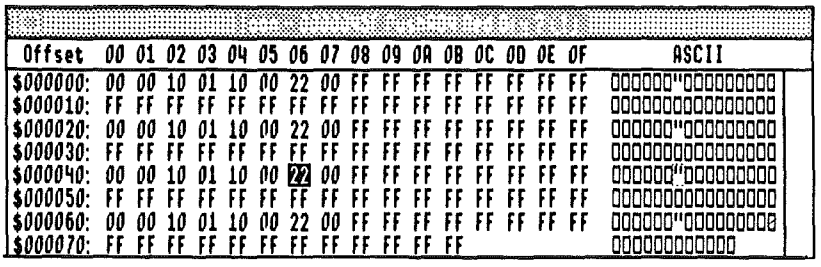


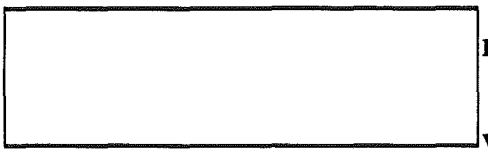
Figure 4-5 The HexEdit Window

you can use the tab key to switch between the two views and the arrow keys to move the selection cursor.

There are two editing modes available in HexEdit - insertion and overstrike. In insertion mode, a vertical line cursor indicates where new data will be inserted. In overstrike mode, the byte being changed is highlighted. The cursor location is indicated in the inactive view as a red frame or vertical line. Changing the data in one view automatically updates the other view. For example, if you change the byte selected in Figure 4-4, located at offset $000046, from 22 to 48, the " character that is enclosed by a box in the ASCII display becomes an F. Similarly, replacing the " with an F changes the 22 in the hex view to 48.

APPENDIX A - SCRIPTBUILDER FIELD DESCRIPTIONS

Basic Numeric Sizes	**Byte**	One Byte (unsigned 0 - 255; signed -128 - 127; hex 0 - FF)
	Word	Two Bytes (unsigned 0 - 65535; signed -32768 - 32767; hex 0 - FFFF)
	Long	Four Bytes (unsigned 0 - 4294901760; signed -2147450880 - 2147450879; hex 0 - FFFFFFFF)

String Types	**Character**	An array of ASCII characters with no delimiters. The size of this unit must be defined externally
	**Pascal String**	An array of characters (0 to 255) preceeded by a length byte
	Word String	An array of characters (0 to 65535) preceeded by a length word.
	C String	An array of characters terminated by a null (00) byte

Numeric Types 	**Unsigned**	A byte, word or long real value
	**Signed**	A byte, word or long value which may have positive or negative value
	**Constant**	A fixed byte, word or long value
	**Hexidecimal**	A byte, word or long value expressed in hexidecimal notation (for example $0F, or in C 0x0F)
	**Parameter Count**	A byte, word or long value denoting the number of fields to follow. A range may be established.

Location Types 	**Point**	Two words containing horizontal and vertical coordinates
	**Rectangle**	Two points (four words) containing the top left and bottom right points of the rectangle's boundry

Choice Types 	**Boolean**	A byte, word or long value denoting true (non-zero) or false (zero)
	**Bitfield Array**	An array of boolean values combined to applied bitwise to a byte, word or long value
	**Enumerated Array**	A value associated with another item (for example, a number associated with a country name)

Special Types	**Resource ID**		A reference to another resource of a specified type in the same resource file or Sys.Resources which may be typed into the box or selected using a list which appears on a mouse down while the cursor is in the triangle corner of the box.
	Long Version		A special type of defined string used in rVersion resources to denote major and minor versions
Control Operators	**Label**	Label	A comment or descriptor to indicate what type of data should be entered into a field
	Switch		Branches to another segment of the script based on a value entered into a script
	Goto		Branches to another fixed segment of the script
	Loop		Creates a subgroup which may be repeated a number of times based on the parameters used in setting it up (for example, an rControlList consists of a loop of Resource IDs terminated by a long null ($00000000).
Conditional Statements	**If Equal**		These conditional tests can be used to determine values or in conjunction with control operators to maintain a variable length context based structure.
	If Not Equal		
	If Less Than		
	If GreaterThan		
	If Less Than or Equal		
	If Greater Than or Equal		

APPENDIX B - APPLE IIGS SYSTEM RESOURCE FORMATS

rIcon **$8001**

word	iconType
word	iconSize
word	iconHeight
word	iconWidth
array	iconImage
array	iconMask

rPicture **$8002**

word	picSCB
rect	picFrame
word	picImage
array	pictureImage

rControlList **$8003**

array	ctlList

rControlTemplate **$8004**

CheckBox $82000000

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
long	titleRef
word	initialValue
long	colorTableRef
block	keyEquivalent

Icon Button $87FF0001

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
long	iconRef
long	titleRef
long	colorTableRef
word	displayMode
block	keyEquivalent

LineEdit $83000000

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
word	maxSize
long	defaultRef

List $89000000

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
word	listSize
word	listView
word	listType
word	listStart
long	listDraw
word	listMemHeight
word	listMemSize
long	listRef
long	colorTableRef

Picture $8D000000

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
long	pictureRef

Pop-Up Menu $87000000

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
word	titleWidth
long	menuRef
word	initialValue
long	colorTableRef

Radio Button $84000000

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
long	titleRef
word	initialValue
long	colorTableRef
block	keyEquivalent

Rectangle $87FF0003

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
word	penHeight
word	penWidth
array	penMask
array	penPattern

Scroll Bar $86000000

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
word	maxSize
word	viewSize
word	initialValue
long	colorTableRef

Simple Button $80000000

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
long	titleRef
long	colorTableRef
block	keyEquivalent

Size Box $88000000

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
long	colorTableRef

StatText $81000000

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
long	textRef
word	textSize
word	just

TextEdit $85000000

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
long	textFlags
rect	indentRect
long	vertBar
word	vertAmount
long	horzBar
word	horzAmount
long	styleRef
word	textDescriptor
long	textRef
long	textLength
long	maxChars
long	maxLines
word	maxCharsPerLine
word	maxHeight
long	colorRef
word	drawMode
long	filterProcPtr

Thermometer $87FF0002

word	pCount
long	ID
rect	rect
long	procRef
word	flag
word	moreFlags
long	refCon
word	value
word	data
long	colorTableRef

rC1InputString **$8005**

word	length
array	stringCharacters

rPString **$8006**

byte	lengthByte
array	stringCharacters

rStringList **$8007**

word	count
array	strings

rMenuBar **$8008**

word	version
word	menuBarFlag
array	menuRefArray

rMenu **$8009**

word	version
word	menuID
word	menuFlag
long	menuTitleRef
array	itemRefArray

rMenuItem **$800A**

word	version
word	itemID
byte	itemChar
byte	itemAltChar
word	itemCheck
word	itemFlag
long	itemTitleRef

rTextForLETextBox2 **$800B**

word	length
array	stringCharacters

rCtlDefProc **$800C**

is a Code Resource

rCtlColorTbl **$800D**

Check Box, Radio Button

word	boxReserved
word	boxNor
word	boxSel
word	boxTitle

Icon Button, Simple Button

word	bttnOutline
word	bttnNorBack
word	bttnSelBack
word	bttnNorText
word	bttnSelText

List

word	listFrameClr
word	listNorTextClr
word	listSelTextClr
word	listNorBackClr
word	listSelBackClr

Pop-Up Menus

word	NorText
word	HiText
word	Outline

Scroll Bar

word	barOutline
word	barNorArrow
word	barSelArrow
word	barArrowBack
word	barNorThumb
word	barSelThumb
word	barPageRgn
word	barInactive

```
    SizeBox
        word        Outline
        word        NorBack
    TextEdit
        word        contentColor
        word        outlineColor
        word        hiliteForeColo
        word        hiliteBackColor
        word        vertColorDescriptor
        long        vertColorRef
        word        horzColorDescriptor
        long        horzColorRef
        word        growColorDescriptor
        long        growColorRef;
    Thermometer
        word        outline
        word        interior
        word        foregnd
        word        backgnd

rWindParam1         $800E
    word        plLength
    word        plFrameBits
    long        plTitle
    long        plRefCon
    rect        plZoom
    long        plColor
    word        plYOrigin
    word        plXOrigin
    word        plDataH
    word        plDataW
    word        plMaxH
    word        plMaxW
    word        plScrollVer
    word        plScrollHor
    word        plPageVer
    word        plPageHor
    long        plInfoText
    word        plInfoHeight
    long        plDefProc
    long        plInfoDraw
    long        plContentDraw
    Rect        plPosition
    long        plPlane
    long        plControlList
    word        plInDesc

rWindParam2         $800F
    word        p2ListID
    long        p2DefProc
    array       p2Data

rWindColor          $8010
    word        frameColor
    word        titleColor
    word        tbarColor
    word        growColor
    word        infoColor

rTextBlock          $8011
    array       stringCharacters

rStyleBlock         $8012
    word        version
    long        rulerListLength
    array       theRulerList
    long        styleListLength
    array       theStyleList
    long        numberOfStyles
    array       theStyles

rToolStartup        $8013
    word        flags
    word        videoMode
    word        resFileID
    long        dPageHandle
    word        numTools
    array       toolArray

rResName            $8014
    word        versNum
    long        nameCount
    array       resNames

rAlertString        $8015
    array       alertString

rText               $8016
    array       stringCharacters

rCodeResource       $8017
    is a code resource.

rCDEVCode           $8018
    is a code resource.

rCDEVFlags          $8019
    word        flags
    byte        enabled
    byte        version
    byte        machine
    byte        reserved
    rect        dataRectangle
    array       name
    array       author
    array       verString

rTwoRects           $801A
    rect        rect1
    rect        rect2

rFileType           $801B
    word        Version
    word        Flags
    word        NumEntries
    word        reserved
    word        IndexRecordSize
    word        OffsetToIdx
    array of
        word        Filetype
        long        Auxtype
        word        flagword

rListRef            $801C
    long        ID
    byte        itemFlag
    array       item

rCString            $801D
    array       stringCharacters

rXCMD               $801E
    is a code resource.

rXFCN               $801F
    is a code resource.

rErrorString        $8020
    array       alertString

rKTransTable        $8021
    array       transTable
    array       deadKeyTable
    array       replacementTable

rWString            $8022
    word        length
    array       stringCharacters

rC1OutputString     $8023
    word        bufferSize
    word        stringLength
    array       stringCharacters

rSoundSample        $8024
    word        format
    word        waveSize
    word        relPitch
    word        stereo
    word        sampleRate
    array       sound

rTERuler            $8025
    word        leftMargin
    word        leftIndent
    word        rightMargin
    word        just
    word        extraLS
    word        flags
    long        userData
    word        tabType
    array       theTabs
    word        tabTerminator

rFSequence          $8026
    reserved by Apple

rCursor             $8027
    word        height
    word        width
    array       image
    array       mask
    word        hotSpotY
    word        hotSpotX
    word        flags
    array       reserved

rItemStruct         $8028
    word        itemFlag2
    long        itemTitleRef
    long        itemIconRef

rVersion            $8029
    long        version
    word        country
    array       name
    array       moreInfo

rComment            $802A
    array       text

rBundle             $802B
    word        version
    word        offsetToDocList
    long        iconID
    long        ID
    long        reserved
    word        count
    array of
        word        oneDocSize
        word        offsetToMatchFlags
        word        numResults
        word        priority
        long        finderPath
        long        reserved
        long        iconID
        long        reserved
        long        smIconID
        long        reserved
        array       description
        long        matchFlags
        array       matchFields

rFinderPath         $802C
    word        version
    word        offsetToPathName
    word        pCount
    long        versionID
    long        reserved
    WString     pathname

rPaletteWindow      $802D
    is a code resource.

rTaggedStrings      $802E
    word        count
    array of
        word        Key
        array       String

rPatternList        $802F
    array of
        pattern     thePattern

rRectList           $C001
    word        count
    array of
        rect        theRect

rPrintRecord        $C002
    array       printRecord

rFont               $C003
    array       font
```

APPENDIX C - RESOURCE DEPENDENCIES

Listing Format-

ParentKind
Child that May Have Dependents
Child that has No Dependents

rControlList	**$8003**
rControlTemplate	$8004
rControlTemplate	**$8004**
rIcon	$8001
rPicture	$8002
rC1InputString	$8005
rPString	$8006
rMenu	$8009
rTextForLETextBox2	$800B
rCtlColorTbl	$800D
rStyleBlock	$8012
rText	$8016
rListRef	$801C
rCString	$801D
rC1OutputString	$8023
rMenuBar	**$8008**
rMenu	$8009
rMenu	**$8009**
rMenuItem	$800A
rMenuItem	**$800A**
rPString	$8006
rItemStruct	$8028
rWindParam1	**$800E**
rControlList	$8003
rControlTemplate	$8004
rPString	$8006
rWindColor	$8010
rItemStruct	**$8028**
rIcon	$8001
rPString	$8006
rBundle	**$802B**
rIcon	$8001

APPENDIX D - RESOURCE MANAGER ERRORS

resForkUsed	$1E01
resBadFormat	$1E02
resNoConverter	$1E03
resNoCurFile	$1E04
resDupID	$1E05
resNotRound	$1E06
resFileNotFound	$1E07
resBadAppID	$1E08
resNoUniqueID	$1E09
resIndexRange	$1E0A
resSysIsOpen	$1E0B
resHasChanged	$1E0C
resDiffConverter	$1E0D
resDiskFull	$1E0E
resInvalidShutDown	$1E0F
resNameNotFound	$1E10
resBadNameVersion	$1E11
resDupStartUp	$1E12
resInvalidTypeOrID	$1E13
resBadData	$1E40
resBadStructure	$1E41
resBadFreeList	$1E42

APPENDIX E - INSTALLATION NOTES

Foundation requires that whatever disk the program is launched from must have enough disk space to store a copy of the resource fork you wish to edit. Foundation creates a directory when first launched named `Foundation.User` to store user preferences and user-created scripts.

If you are using a system without a hard drive or network volume, create a working copy of Foundation by duplcating the program disk and then deleting all files on the working disk but `Foundation` and the directory `Foundation.Edit`. On all other configurations, place the file `Foundation` and the directory `Foundation.Edit` in the same directory.

The `Foundation.User` directory is created in the same directory as the Foundation file except when the program is launched from an AppleShare (NOT Macintosh System 7 Personal File Sharing) volume. In this case, the directory is created in the appropriate User directory.