For ResEdit 2.0b2

ResEdit Reference

Draft

Developer Technical Publications

APPLE COMPUTER, INC.

20525 Mariani Avenue
Cupertino, CA 95014-6299
(408) 996-1010

Simultaneously published in the United States and Canada.

Limited Warranty on Media and Replacement

If you discover physical defects in the manual or in the media on which a software product is distributed, APDA will replace the media or manual at no charge to you provided you return the item to be replaced with proof of purchase to APDA.

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, **APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

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

Contents

Figures and tables

Preface

ResEdit™, an extensible standalone resource editor for the Macintosh® computer, is a powerful tool you can use to speed your software development process and to create icons, menus, and other resources for Macintosh programs and files. This manual is a complete reference to ResEdit that includes introductions to the various resource type editors included in the program, and a discussion of the framework that is provided so that you can extend the capabilities of the program by adding your own resource pickers and editors.

Prerequisites

To run ResEdit 2.0, the system you use must have at least 128 KB of ROM and at least 1 megabyte of memory.;

ResEdit 2.0 works with system software version 5.0 and later. ResEdit is compatible with (but does not require) 32-bit QuickDraw™.

What this manual contains

Chapter 1 introduces the concepts behind ResEdit, starting with an overview of Macintosh resources. Chapter 2 tells you about the user interface. Chapter 3 discusses the individual editors that are built into the program. Chapter 4 then describes template editing, and Chapter 5 tells you how to build your own templates. Chapter 6 is a "hints and kinks" area in which we include useful information that will help you make efficient use of ResEdit. Chapter 7 describes the programmatic interface to ResEdit and tells you what you need to know in order to write your own picker and editor. Appendix A describes the inner workings of the 'KCHR' editor, Appendix B describes the inner workings of the 'BNDL' resource, Appendix C lists a number of extant resource types, and Appendix D is a chart of the regular Macintosh character set.

How to use this manual

If you have used previous versions of ResEdit, you will probably want to take a quick look at Chapter 2, which describes the user interface in some detail, specifically because the interface has been changed extensively in version 2.0.

If you have never used ResEdit, you should probably read Chapters 1 and 2 and look over the rest of the book. Use the program for a while, and then look at the book again. It will probably make a lot more sense after you've actually played with ResEdit.

Conventions used in this book

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

◆ *Note:* A note like this contains information that is interesting but not essential for an understanding of the main text.

△ **Important** A note like this contains information that is essential. △

▲ **Warning** Warnings like this indicate potential problems. ▲

This manual uses `courier` type to represent code fragments and the names of procedures.

Graphics

Most of the artwork in this book is taken directly from Macintosh screens. Some illustrations show a condensed version of the screen with a sequence of windows or some particular feature (such as a menu) evident. Others show only an active window, or an alert or dialog box.

Where to get information

Apple technical books published by Addison-Wesley, such as *Inside Macintosh*, are available at commercial bookstores. Books and manuals published by Apple are available through APDA, the Apple Programmers and Developers Association, at the address listed below. Technical notes and other materials of interest to Macintosh application developers are also available from APDA.

About APDA

APDA provides a wide range of technical products and documentation, from Apple and other suppliers, for programmers and developers who work on Apple equipment. You can contact them as follows.

APDA
Apple Computer, Inc.
20525 Mariani Avenue, M/S 33-G
Cupertino, CA 95014–6299

Telephone: 1-800-282-APDA or 1-800-282-2732 if you are inside the United States; in Canada, 1-800-637-0029; elsewhere in the world, 01-408-562-3910.
Fax: 408-562-3971 Telex: 171-576 AppleLink: DEV.CHANNELS

About Developer Programs

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:

Apple Developer Programs
Apple Computer, Inc.
20525 Mariani Avenue, M/S 75-2C
Cupertino, CA 95014–6299

Chapter 1 **ResEdit Overview**

This chapter introduces the concept of resources as they are handled on the Macintosh® computer, and introduces ResEdit™, an interactive, graphically based application for manipulating resources in Macintosh files. Some Macintosh files don't contain any resources, but all applications and most of the System Folder files do.

Resources

One of the ways in which the Macintosh is different from other computers is its handling of *resources* (typefaces, icons, dialog boxes, and so on). In the Macintosh, resources are distinct from data (for example, the text in a word-processing file). The Macintosh does not insist on keeping resources in a central pool; they may be placed in any file.

In most computers, a file consists of a set of bytes, perhaps beginning with a header that contains some information about the structure of the data contained in the file, and possibly ending with some sort of trailer; in any case, the file is one set of bytes. The Macintosh has, instead, a file structure that is designed to include two sets of bytes, a *data fork* and a *resource fork*. Any file may contain only a data fork, only a resource fork, or both. While a plain HyperCard stack, for example, has only data in it, people commonly add icons and sounds to their stacks, creating resource forks for those stacks in the process.

Resources are classified by type. Each type has its own name, which consists of exactly four characters. Any characters in the Macintosh character set can occur in resource type names, even unprintable ones, but typically they consist of lower- and uppercase letters, numerals, punctuation marks, space, and Option-space. Resource type names are shown here with single straight quotation marks around them (for example, 'itl0'). If you see a name that appears to be shorter, the empty slots are probably filled with spaces (for example, 'snd '). Some resource types are named and described in Appendix C. There are many different types of resources, and you can create your own resource types with ResEdit if you don't find the type you need.

- *Note:* Apple Computer, Inc., reserves all names that don't contain any uppercase letters. Any combination with at least one uppercase letter in it is yours to use, though it is a good idea to avoid using any type name that someone else has already used that you know of.

Another feature of this system is that code is regarded as a resource. It even has its own resource type name (very straightforwardly, 'CODE'). Any application, then, must have a resource fork, which is where its code resides, along with various other resources, such as menus.

ResEdit lets you copy and paste all resource types, and lets you edit many of them. ('NFNT' is an exception, and is discussed briefly in the section on 'FONT' editing, in Chapter 3.) ResEdit actually includes a number of different resource editors: There is a **general resource editor** for editing any resource in hexadecimal and ASCII formats, and there are individual resource editors for various specific resource types. There is also a **template editor,** which lets you edit some kinds of resources in a dialog box format, with fields that you can fill in as appropriate. There are predefined templates for quite a few resources already built into ResEdit, and you can create others. For further information on template editing and on generating your own templates, see Chapters 4 and 5.

New and changed resource types supported by ResEdit 2.0

The 'cicn' color icon resource type is up to 8 bits deep and contains its own color lookup information. It defaults to a size of 32 x 32 pixels, though both its height and its width can be changed independently to be anything from 8 to 80 pixels. This icon resource includes a monochrome version and mask. ResEdit 2.0 includes an editor for 'cicn' resources.

Finder icons for system software version 7.0 occur in 6 variants, including the old 'ICN#' and the new 'ics#' types, as well as the new 4- and 8-bit small and large color icons ('ics4', 'ics8', 'icl4', and 'icl8'). A comprehensive editor in ResEdit 2.0 lets you deal with each set of Finder icons as a coherent group.

ResEdit 2.0 includes an editor for bundles, resource type 'BNDL' (bundles also involve other resources, as described in Appendix B), and an editor for menus, resource types 'MENU', 'cmnu', 'CMNU', and 'mctb'. These editors are discussed in Chapter 3.

Resource categories in ResEdit

ResEdit behaves as if there were three kinds or categories of resources on the Macintosh.

Resources of the first kind are accessed with individual pickers and edited with individual editors. These resources and their editors are described in some detail in Chapter 3. Several of these resources ('CURS', 'FONT', 'ICON', 'PAT ', and so on) are in some sense pictorial. All of the pictorial resources are edited with bit editors, which are discussed in Chapter 3.

Resources of the second kind are edited as templates. That is, if you open a resource of this kind, you are presented with a dialog box in which there are various labeled fields. You can change the contents of the fields. Information on existing templates and on generating your own templates appears in Chapters 4 and 5, and an example of template editing appears in Chapter 6.

Resources of the third kind are edited with the hexadecimal editor, unless you write your own templates or editors for them.

Uses

ResEdit is especially useful for creating and changing graphic resources such as dialog boxes and icons. For example, you can use ResEdit to try out different formats and presentations of resources in the process of putting together a quick prototype of a user interface. Anyone can quickly learn to use ResEdit for translating resources into languages other than English without having to recompile programs. You can use ResEdit to modify a program's resources at any stage in the process of program development. ResEdit is also useful for modifying the 'LAYO' (desktop layout control) resource in a copy of the Finder™ so that you can reconfigure some aspects of the desktop display. See Chapter 6 for more details about the 'LAYO' resource.

Extensibility

A key feature of ResEdit is its extensibility. Because it can't anticipate the formats of all the different types of resources that you may use, ResEdit is designed so that you can teach it to recognize and parse new resource types.

There are two ways that you can extend ResEdit to handle new types:

- You can create templates for your own resource types. ResEdit lets you edit most resource types by filling in the fields of a dialog box; this is the way you edit the Finder's desktop layout control resource, for example. The ordering of the items in these dialog boxes is determined by a template in ResEdit's resource file, and you can add templates to ResEdit or to the ResEdit preferences file yourself to edit new resource types. Resource templates are described in Chapters 4 and 5, and the desktop layout control resource is discussed in some detail in Chapter 6.

- You can program your own special-purpose **resource picker** or **editor** (or both) and add it to either ResEdit or to the ResEdit Preferences file. (The *resource picker* is the code that displays all the resources of one type in the resource type window. The *editor* is the code that displays and allows you to edit a particular resource. These pieces of code are separate from the main code of ResEdit.) A set of Pascal or C routines, called ResEd, is available for this purpose—see Chapter 7 for information. The advantage of adding your code to the ResEdit Preferences file rather than to ResEdit itself is that doing so facilitates updating to new versions of ResEdit as they become available.

The resource development cycle

ResEdit is often used with Macintosh Programmer's Workshop (MPW®) and other program development systems. Once you have created or modified a resource with ResEdit, you can use the MPW resource decompiler, DeRez, to convert the resource to a textual representation that can be processed by the resource compiler, Rez. You can then add comments to this text file or otherwise modify it with the MPW Shell or another text editor. Rez and DeRez are fully described in the *Macintosh Programmer's Workshop Reference* (*MPW Reference*). It is not necessary to use Rez or DeRez unless you have some specific need or desire to modify or comment the code that DeRez produces; the resources generated by ResEdit are, in general, entirely acceptable.

Chapter 2 **Getting Started**

If you are new to ResEdit, you will want to proceed with some caution, as ResEdit is quite powerful and can easily damage or destroy your files. If you are accustomed to earlier versions of ResEdit, you will notice that the user interface has been extensively changed and now conforms more closely to the guidelines established by Apple Computer, Inc.

Invoking ResEdit

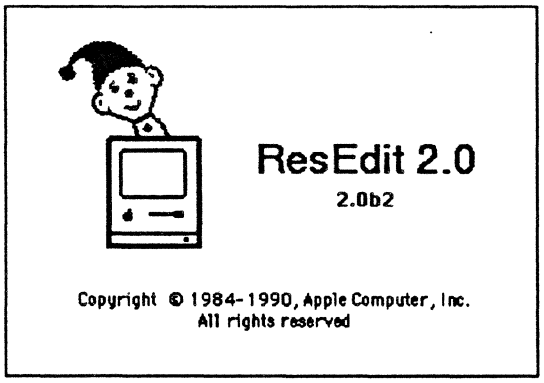
ResEdit is a regular application, so if you are in the Finder or in HyperCard you can start it up just as you would any other application. If you are using MPW, you can start ResEdit by entering either of these commands in the MPW Shell:

```
ResEdit
ResEdit file1 file2 ...
```

The latter command causes ResEdit to open the named files automatically.

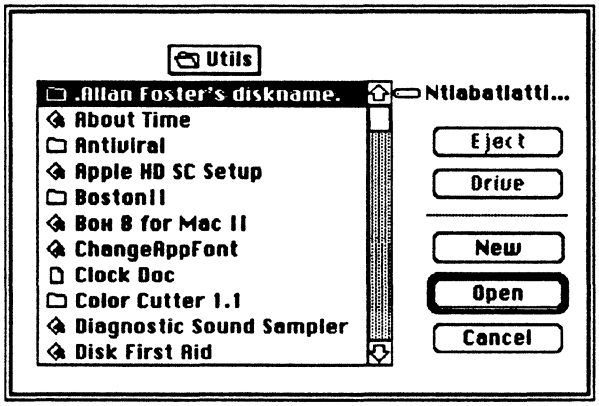
When ResEdit first starts up, it displays an animated "splash screen." Figure 2-1 shows one of the stages of this animation.

■ **Figure 2-1** Splash screen



The animation continues until you click the mouse anywhere or press any key. If you click the mouse or press an unmodified key ResEdit puts up a dialog box, shown in Figure 2-2, that lets you create a new file or open an existing one. If you press a command-key combination, the splash screen is dismissed and ResEdit performs the action you have requested. This is especially useful for command keys assigned to the Open Special menu, described in this chapter. You can, if you wish, use the Preferences command on the File menu to choose not to have ResEdit put up the dialog box.

■ **Figure 2-2** ResEdit file open dialog



You can select a filename by clicking it or by typing one or more characters of the filename.

Working with files

ResEdit provides facilities to let you open files, create new files, create resources, move and edit them, and perform two levels of verification on them.

Resource checking

Sometimes a resource file gets corrupted. This is typically the result of a crash occurring while the file is being updated. In the past, ResEdit would occasionally crash when you tried to open a damaged file with it. Version 2.0 of ResEdit provides resource file checking facilities to help avoid crashes and to minimize loss of data. The checking facility does not detect corrupted individual resources; it bases its tests on the file's resource map.

When you open a file, ResEdit performs a partial resource check on it. This test verifies only that the resource map is located after the end of the resource data area, and that the header, data, and map do not extend beyond the EOF of the resource fork. If the file does not pass these initial tests, a full test is automatically performed. If you choose "Verify files when they are opened" in the Preferences dialog, ResEdit performs a full test whenever you open a file.

If you want to invoke the full test yourself, choose Verify Resource File from the File menu.

In order to perform a full resource check, ResEdit walks through the entire resource map and verifies that the type list, the reference lists, and the name list are consistent, that all resource data areas can be located, and that they do not exceed the available file size. It also checks for duplicate types, and for duplicate ID numbers within each type. ResEdit has several techniques for locating the resource map, the existence and location of which is critical to the process of recovering damaged resource files.

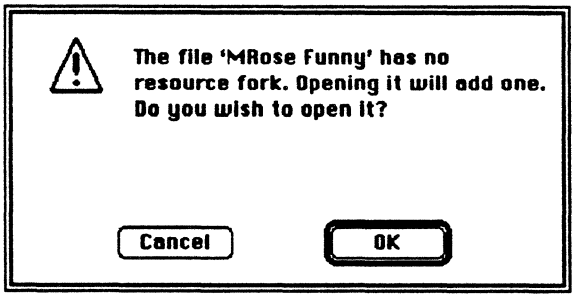
If damage is discovered, the user is offered a repair option. This procedure does not change the damaged file. Instead, ResEdit creates a new file, extracts all the resources it can find in the damaged file, and copies them to the new file. It then renames the old file (with an extension of "(damaged)". ResEdit also presents the user with status information about the resources that were extracted.

There is one exception to the rule that the damaged file is not "touched" minor damage occurs whenever a resource file is not properly closed. ResEdit repairs this damage without asking the user's permission. (The actual process involved is quite simple: ResEdit opens the file using the Resource Manager, calls the `UpdateResFile` routine to rewrite the resource map, and closes the file.) After performing the repair, it presents an alert to the user.

Opening a file

To list the resource types in a file, select and open the filename from the list in the file open dialog. If you try to open a file that does not have a resource fork, ResEdit displays a dialog box, shown in Figure 2-3, that asks you whether you want to open the file anyway. If you permit it to open the file, ResEdit extends the file by creating a resource fork in it.

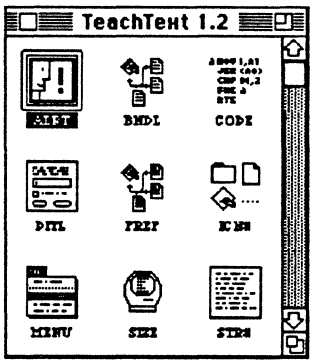
■ **Figure 2-3** Add resource fork alert



▲ **Warning** You can edit any file shown in the window, including the System file and ResEdit itself, though there are some restrictions (the Finder and the Desktop File cannot be opened by ResEdit under MultiFinder™, for example). It's dangerous, though, to edit a file that's currently in use. In general, it is much wiser to edit a duplicate instead of the file itself. ▲

When you open a file, a **file window** appears. This window displays a pictorial list of all the resource types in that file (See Figure 2-4), unless you choose "by Name" from the View menu (See Figure 2-18). If you do choose to view the resource list by name, you can also choose to show the total size of each resource type.

■ **Figure 2-4** A ResEdit 2.0 file window



When a file window is the active window, you can create new resource types, copy or delete existing resources, and paste resources from other files. Here, operations are performed on sets of resources. For example, selecting the resource type 'ALRT' in a file causes all resources of type 'ALRT' in that file to be selected as a group. Any operation you then perform on that group affects all 'ALRT' resources in the file. To select more than one resource type, hold down the Command key while clicking the individual items or click an item at the beginning of the range you want to select, hold down the Shift key, and click the item at the end of the range. The Shift key allows you to select the items in a rectangular area. You can then continue to select or deselect individual resource types with the Command key. (These techniques will also work within an open resource type for selecting individual resources.)

- *Note:* Many applications put more than one resource type at a time into the scrap when Copy is chosen. For example, when an object is copied in MacDraw®, an 'MDPL' resource and a 'PICT' resource are put into the scrap. When you paste into the file window in ResEdit, all resources that are present are pasted.

- *Note:* You can no longer use ResEdit to delete files; also, ResEdit does not manipulate or read data forks (this means, for example, that it cannot copy them).

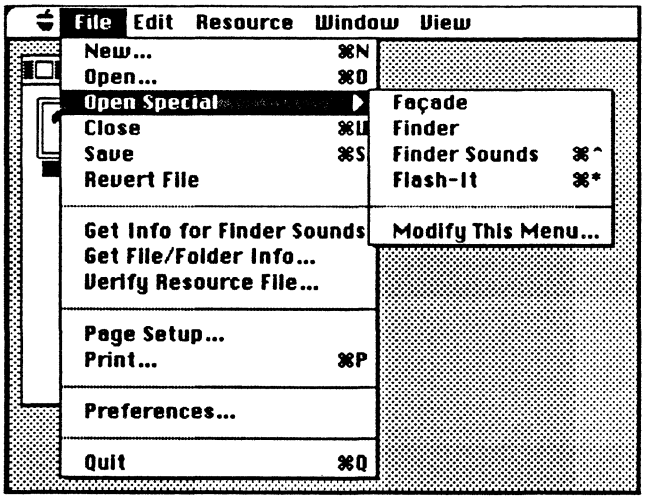
Menus in ResEdit

The structure of menus in ResEdit has been changed with the 2.0 release. There are five main menus discussed here (File, Edit, Resource, Window, View), and special menus for particular resources that are discussed in the sections on editing those resources, in Chapter 3.

The File menu

Figure 2-5 shows the File menu.

■ **Figure 2-5** File menu



The File menu commands act as follows:

New...	Brings up the new file dialog box.
Open...	Brings up the file open dialog box similar to the one shown in Figure 2-2, but without a New button.
Open Special	Allows you to open files quickly. The Modify This Menu command, which always appears at the bottom of the submenu, brings up the dialog box shown in Figure 2-6, which allows you to add and remove files and command keys.

Close — Closes the currently active window. (Using this command has the same effect as clicking the close box.)

Save — Saves the currently active file, if there is one.

Revert file — Restores the currently active file, if there is one, to the last version you saved.

Get Info for This File

When no file is open this command is gray and cannot be used. When a file is open the words "This File" are replaced by the filename, and this command is enabled. It displays file information and allows you to change it. (See Figure 2-7.)

Get File/Folder Info...

Displays file or folder information and allows you to change it. Figure 2-7 shows a File Info window as it appears under system software version 6.0. Figure 2-8 is a Folder Info window, also for system software version 6.0.

Verify Resource File...

Allows you to check the resource map of a file you specify.

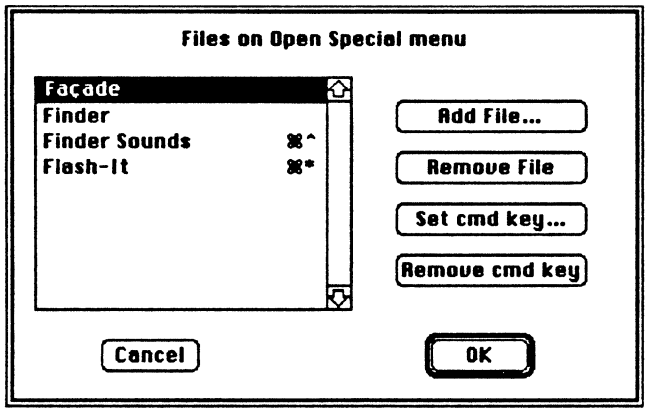
Page Setup... — Brings up the page setup dialog box.

Print... — Allows you to print from almost any picker or editor. When no files are open, it is gray and cannot be used.

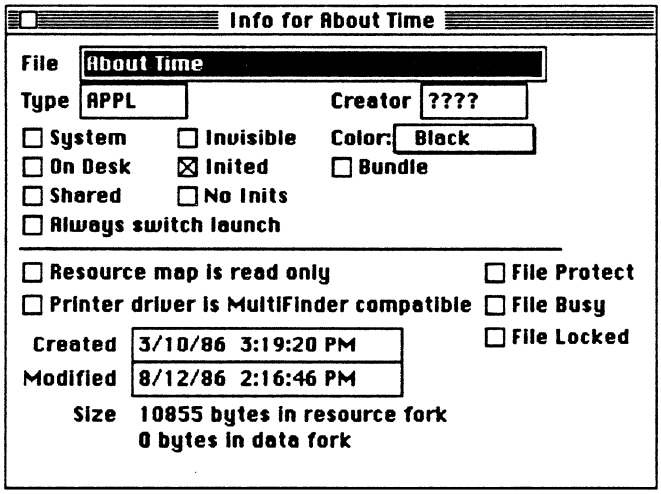
Preferences... — Brings up the dialog box shown in Figure 2-9. This lets you specify whether you want ResEdit to start up with a file open dialog, whether you want to be warned if you attempt to open the System file or ResEdit itself, whether you want ResEdit to perform a verify operation on files when you open them, and also allows you to set the sizes of type picker and resource picker windows.

Quit — Quits ResEdit and returns to the Finder (or the MPW Shell, HyperCard, or whatever program launched ResEdit).

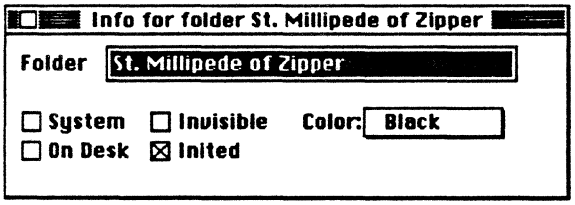
■ **Figure 2-6** Open Special dialog box



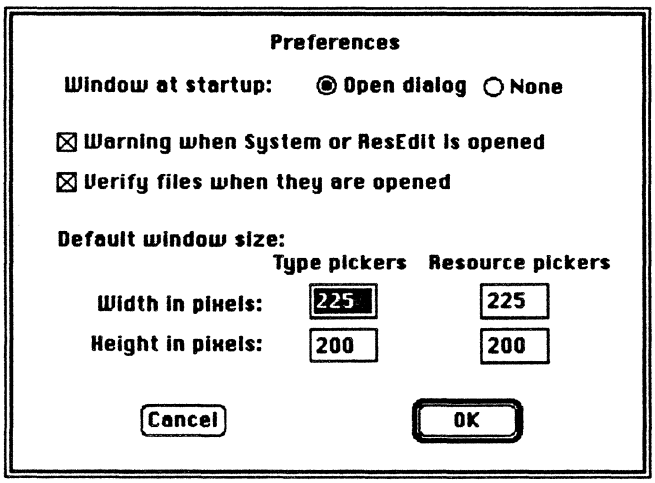
■ **Figure 2-7** A File Info window



■ **Figure 2-8** A Folder Info window



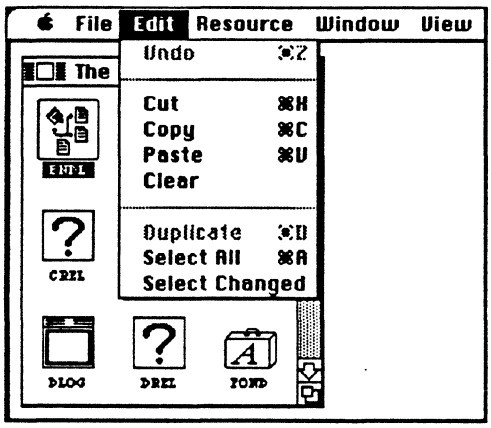
■ **Figure 2-9** Preferences dialog box



The Edit menu

Figure 2-10 shows the Edit menu. It has only one unusual feature, the Select Changed command on the last line. This command allows you to select only those items that have been changed since the last time you saved the file you are working on.

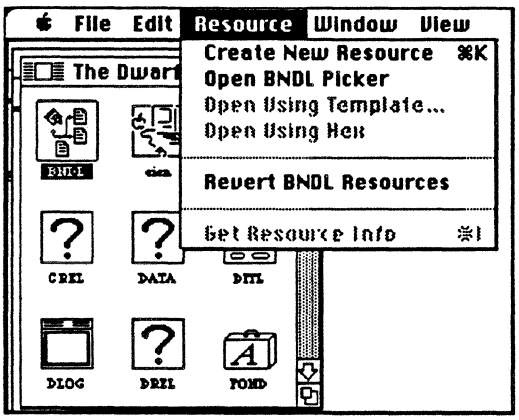
■ **Figure 2-10** Edit menu



The Resource menu

The Resource menu is configured to provide the commands appropriate for the frontmost window. The same items are always present on the menu, but they have slightly different meanings, depending on the context. The wording of the items on the menu always tells you what they do when you choose them. Figure 2-11 shows the Resource menu with a resource type picker open and the 'BNDL' type selected.

■ **Figure 2-11** The Resource menu for 'BNDL'



The Create New Resource command lets you create any resource type. The Open Picker command invokes a picker for the particular kind of resource that is selected. This is reflected in its name, which includes the name of the selected resource type. At this level, the only other command you can use is the Revert Resources command, which takes the resources back to the last saved version. If you have made changes in individual resources of the selected type since the last time you saved the file, you can undo those changes here.

Figure 2-12 shows the Resource menu again, this time with a resource picker open. Note that it is now possible to open a resource with a resource editor or template (if one is available) or with the hexadecimal editor.

■ **Figure 2-12** The Resource menu with a picker open

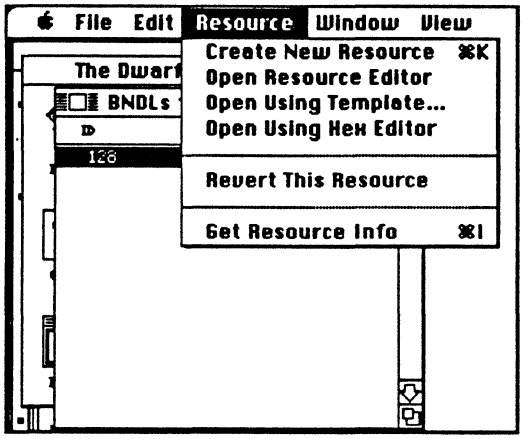
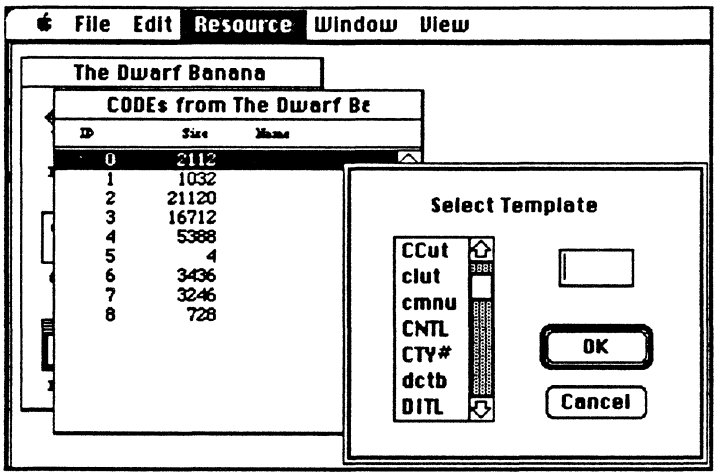


Figure 2-13 shows the result of attempting to use the Open Using Template command on a 'CODE' resource. There is, in fact, no template for resources of this type. It is generally not useful to open a resource of one type with a template for a resource of a different type.

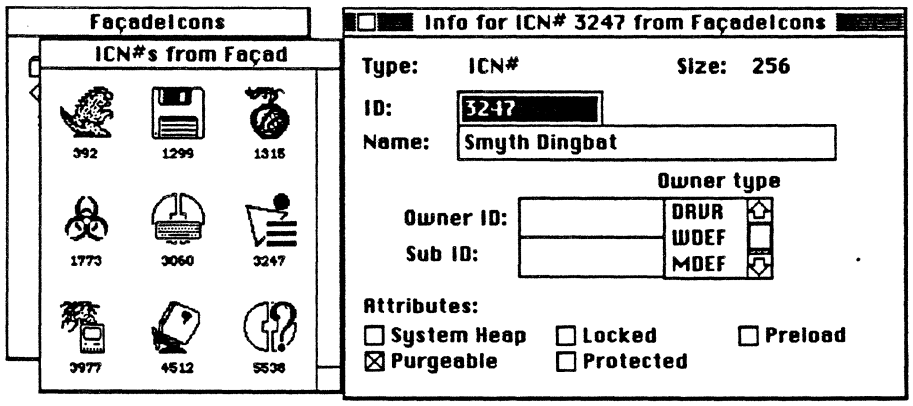
■ **Figure 2-13** There is no template for 'CODE' resources



It is also possible to get information on the selected resource. Figure 2-14 shows the Get Info window for a resource of type 'ICN#'. This dialog lets you change the name and ID number of the resource, and select or deselect some of its attributes.

- **System Heap:** If this attribute is set, the resource is placed in the System heap unless it is too large to fit. In that case, the resource is placed in the Application heap, as if the box were not checked. This attribute should not be set for an Application's resources.
- **Purgeable:** If this attribute is set, the resource can be purged from memory if more room is needed. It is typically a good idea to set this attribute, but there are exceptions.
- **Locked:** If this attribute is set, the resource is locked in place in the heap, and cannot be moved. This attribute overrides the "Purgeable" attribute.
- **Protected:** If this attribute is set, the Resource Manager cannot change the name or ID number of the resource, modify its contents, or remove the resource from the file that contains it. The Toolbox routine that sets these attributes can be called, however, to unset this one.
- **Preload:** Setting this attribute causes the Resource Manager to load the resource into memory immediately after opening the resource file.

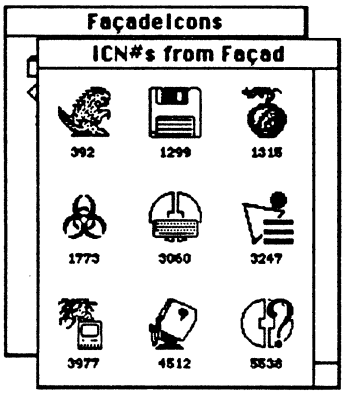
- **Figure 2-14** An 'ICN#' Get Info window



Opening a resource type produces a window that lists each resource of that type in the file. The list is generated by a resource picker, and will take different forms, depending on the particular resource picker that is displaying it. The general resource picker displays the resources by type, name, ID number, or order in the file; pickers for specific resource types generate displays that are appropriate for their type. Figure 2-15 shows a picker for the 'ICN#' resource type.

You can also write your own pickers. For more information, see Chapter 7.

■ **Figure 2-15** A resource type window (with custom picker)



When a resource type window is the active window, the Edit menu commands have the following effects:

Undo	Not usable.
Cut	Removes the resources that are selected, placing them in the ResEdit scrap. If only one resource is selected, it is placed in the clipboard.
Copy	Copies all the resources that are selected into the ResEdit scrap. If only one resource is selected, it is copied to the clipboard.
Paste	Copies the resources from the ResEdit scrap (or from the clipboard) into the resource type window.

◆ *Note:* Only resources of the currently open type are copied into the resource type window.

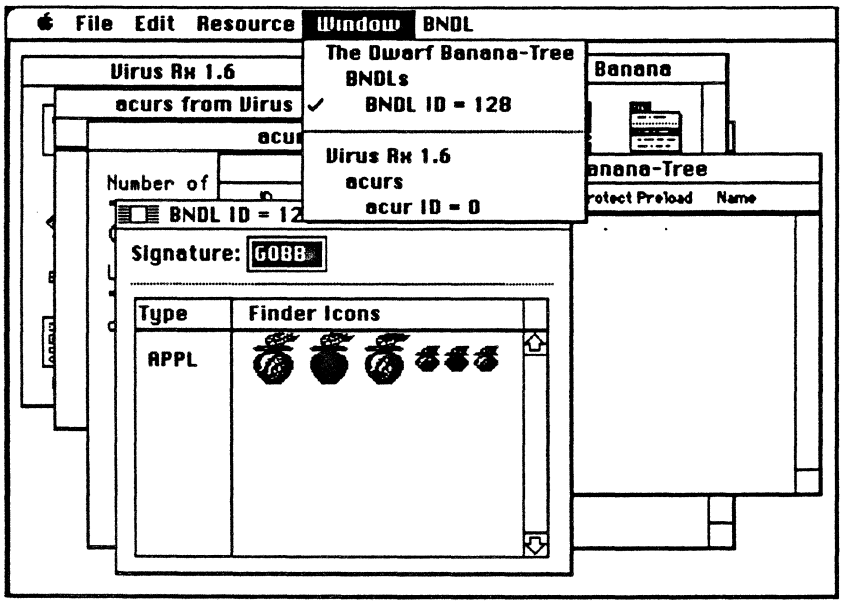
Clear	Removes the resources that are selected without placing them in the ResEdit scrap.
Duplicate	Creates a duplicate of the selected resources and assigns a unique resource ID number to each new resource.

When you choose Open Using Template from the Resource menu, a list of templates is displayed, and you can pick the one you want to use.

The Window menu

The Window menu, shown in Figure 2-16, gives you an overview of what windows are currently open, and indicates the currently active window with a checkmark. It also lets you select a new current window. Note that the Window menu is sorted not by window depth, but by file.

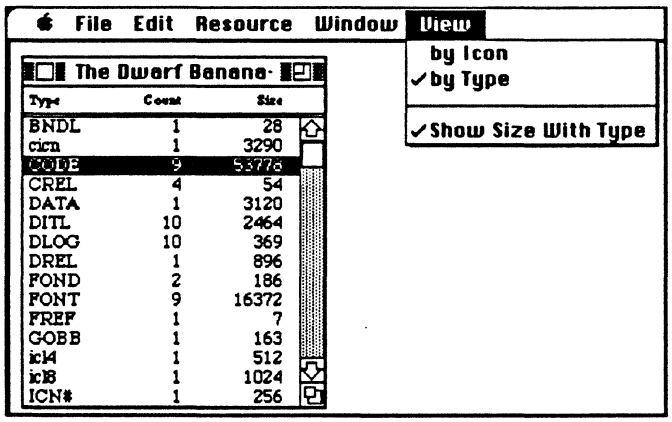
■ **Figure 2-16** The Window menu



The View menu

The View menu is configured to match the frontmost window. When a file window is the currently active window, the View menu lets you show the resource types in a file by Icon or type name, and if you show them by type, it lets you show the size of each type. (That is, the sum of the sizes of all resources within the type.) See Figure 2-17.

■ **Figure 2-17** The View menu and a ResEdit 2.0 file window

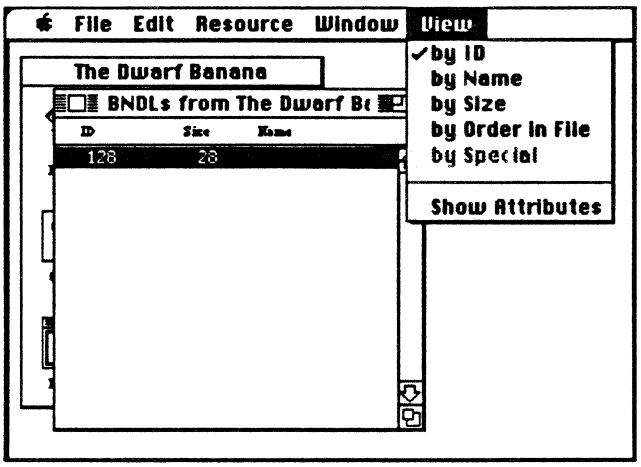


When a resource type window is the currently active window, the View menu lets you choose among several viewing styles (see Figure 2-18), and lets you show some attributes for each resource when you are viewing by ID, Name, Size, or Order in File (See Figure 2-19). Attributes cannot be edited in this view, only displayed.

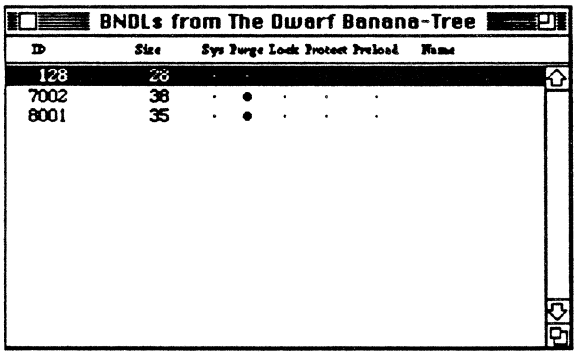
For some resources, the "By Special" line that is grayed out in Figure 2-18 is changed to a type-specific alternate (for example, "By cicn") . Attributes cannot be displayed in the special views.

When an individual resource is open, the View menu is not shown.

■ **Figure 2-18** The View menu and a resource type window



■ **Figure 2-19** Showing type attributes



Resource ID numbers

Within a given resource type, resource ID numbers must be unique. Resources can, in general, have any ID number between –32768 and +32767, but you should be aware of the following restrictions which apply to most resources:

- ID numbers from –32768 to –16385 are reserved. Do not use them!
- ID numbers from –16384 to –1 are used for system resources that are owned by other system resources. For example, a dialog box used by a desk accessory (the desk accessory is, itself, a resource of type 'DRVR') would have a number in this range.
- ID numbers from 0 to 127 are used for system resources.
- ID numbers from 128 to 32767 are available to you for your uses.

Some system resources own others. The "owner" contains code that reads the "owned" resource into memory. For example, desk accessories can have their own patterns, strings, and so on. Please see Chapter 5 of *Inside Macintosh,* Volume I, for more information.

Fonts constitute a special case. For information about fonts, see the section on 'FONT' resources in Chapter 3.

Chapter 3 **Editing Individual Resources**

Some of the ResEdit resource editors are discussed in this chapter. The use of the editors not discussed here should be apparent when you run them. For information on editing template resources, please see Chapter 4.

Starting an editor

To open an editor for a particular resource in a file, first open the picker for the resource type. To do this, either double-click the resource type name or select it and choose Open Picker from the Resource menu. (The command will actually name the resource type. For example, Open BNDL Picker.) Then doubleclick an individual resource, or select it and choose Open Resource Editor from the Resource menu. When an editor is invoked, one or more auxiliary menus may appear, depending on the type of resource you're editing. Some editors, such as the 'DITL' editor, allow you to open additional editors for the elements within the resource. The editors vary in their appearance and function, as explained in this chapter.

If you choose Open Using Template from the Resource menu or hold down the Option and Command keys while opening a resource, a list of templates is displayed. You may then select the template that is appropriate for the resource you are opening. For more information on editing with templates, see Chapter 4.

Bit editors

Pictorial resource types are edited with a bit or pixel editor; ResEdit 2.0 has two major types of bit editor.

Monochrome editors

In the editors for strictly black-and-white resources (for example, the icons commonly used in HyperCard), the cursor acts like the pencil tool in MacPaint.

Holding down the Shift key allows you to use the marquee tool. To make a selection, hold down the Shift key while you drag. To move a selection you've made, Shift-drag. Remember that you must continue to hold down the Shift key; otherwise, your next mouse-click will turn off the marquee and invert whatever pixel the mouse is on. You can also cut, copy, and paste the selections you've made.

- *Note:* If you try to paste more bits than there is room for in the resource (for example, if you try to paste a 40-by-40-bit area from a paint program into, say, an 'ICON', which can only hold a 32-by-32-bit area), ResEdit pastes the selection centered into the active area, and the boundary of the selection will be outside the active area of the editing window. You can shift-drag to reposition the selection.

The 'FONT' editor, discussed in detail later in this chapter, is also a bit editor, but it has a palette with several tools, the use of which is familiar from common paint programs, rather than just the pencil and the marquee.

Color editors

ResEdit 2.0 includes two new bit editors that make use of color when it is available. These are the color icon ('cicn') editor and the Finder icon editor. These editors (and the bundle and menu editors, which also display in color when it is available) have some characteristics that are slightly different from those of the monochrome editors; for instance, when you open, say, a 'cicn' resource, the editor window is placed on the deepest screen. If you have two monitors, one of which is set to black and white and the other is set to sixteen gray levels, the editing window is opened on the monitor that can display gray. When you use ResEdit on systems with 24-bit monitors, you will probably need to increase the Application Memory size beyond the default 512 KB in order to avoid out-of-memory warnings.

The marquee tool is explicitly available on the tool palette in the color icon editors. Holding down the Shift key does not activate the marquee tool in these editors.

Both these bit editors and the other two color editors, for bundles and menus, are discussed in more detail in this chapter.

Using the hexadecimal editor

The hexadecimal resource editor is invoked if you hold down the Option key while opening a resource or choose Open Using Hex Editor from the Resource menu. This editor allows you to edit the resource as hexadecimal or ASCII data. The hex editor can edit resources larger than 255 KB. If a resource is between 256 KB and 511KB in size, each click in the up or down scroll arrow scrolls two lines; if between 512 KB and 767 KB, each click scrolls three lines, and so on. (The scroll bars keep track of position with an integer, which is a single byte, and thus is limited to values between 0 and 255.)

If you enter hexadecimal text when you are using this editor, the editor maintains byte alignment of the incoming data. Thus, if you type `2` into an empty byte, the editor displays `02`. If you then type `A`, the editor displays `2A`.

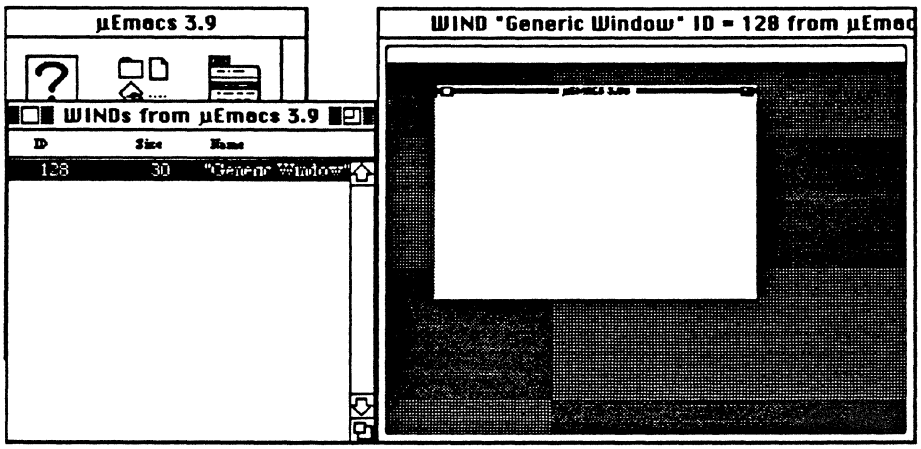
The hex editor has a Search menu. It allows you to search for the occurrence of a pattern in the resource being displayed and allows you to enter the pattern in either hexadecimal or Macintosh character set notation, the latter being loosely described as ASCII, though it is actually considerably larger than the true ASCII set. See Appendix D for a chart of the Macintosh character set. The hex editor also allows you to move to a specified offset from the beginning of the resource you're editing.

'WIND' resources

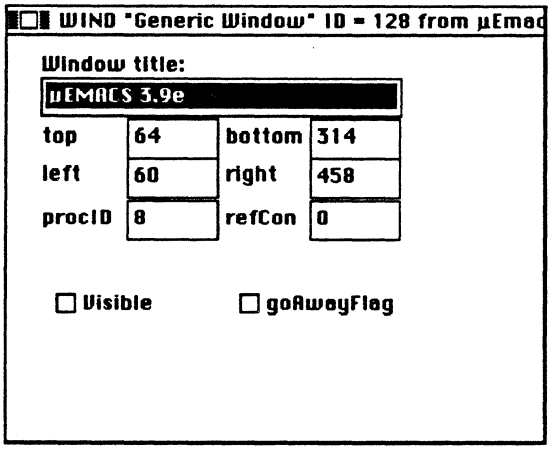
A 'WIND' resource defines a window on the screen. When you open a 'WIND' resource, ResEdit displays a small picture of the screen with the window shown to scale, in its usual size and location. It also presents a special menu with the title WIND. This menu has only one item, Display as Text. You can size the window by using its lower-right corner. You can move the window by clicking anywhere in it, except in its lower-right corner, and dragging the window to where you want it. Moving or sizing a window changes the default values when the window is actually displayed. When the window appears on the screen, its name may be displayed. If the name is displayed, it shows up as a title, in the title bar. To change the name of the window, choose Display as Text from the WIND menu. The text version of a window is shown in Figure 3-2.

Figure 3-1 shows a 'WIND' resource open for editing. Notice the white area across the top of the window in the figure. The white area represents the space that is taken up by the menu bar when the window is actually displayed on the screen.

■ **Figure 3-1** Editing a 'WIND' resource



■ **Figure 3-2** 'WIND' resource displayed as text



'ALRT' and 'DLOG' resources

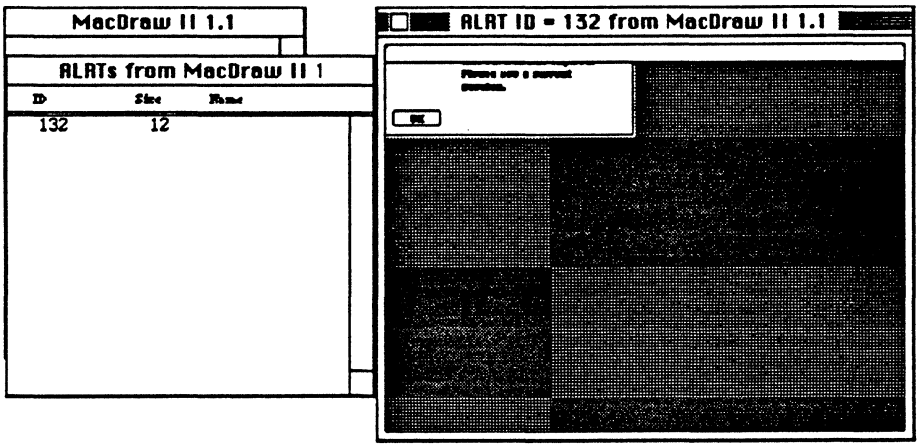
'ALRT' and 'DLOG' resources display dialog boxes on the screen. Editing them is much like editing 'WIND' resources, except that the corresponding 'DITL' resource is automatically opened if you double-click on the picture of the dialog box after opening the resource. (See the next section.) When you display an individual 'ALRT' or 'DLOG' resource, a corresponding menu appears. It has only one item, Display as Text. You can change the resource ID of the associated 'DITL' in the text view, as shown in Figure 3-5.

Figure 3-3 shows an 'ALRT' open for editing. You can see the ALRT menu title in the menu bar. Note the white area at the top of the window, just under the words *Alert ID = 132 from MacDraw II 1.1*; this space is where the menu bar appears when the alert box is displayed on the screen.

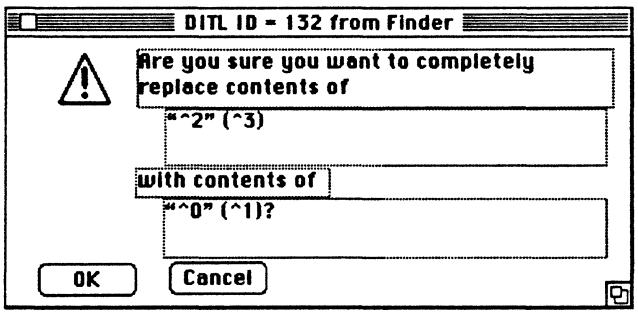
- *Note:* The first item in the 'DITL' associated with any 'ALRT' must be a button. The system has no way of telling what is where, so it always regards the first item as a button. In the alternate view of the 'ALRT', you can specify either item 1 or item 2 as the default. If item 1 is the default, of course, item 2 need not be a button. There is an informal convention in Macintosh programming that item 1 is the "OK" button, and item 2 is the cancel button if there is a cancel button.

There are four special items that you can put into static text in a 'DITL' item or into a 'STR#' resource. They are built of a caret (^) followed by a number from 0 to 3. Each of these refers to one of the items in a global array named DAStrings, maintained by the Dialog Manager. An occurrence of one of these causes the contents of the corresponding entry in that array to be substituted via a *ParamText* call when the resource is displayed. An example of a 'DITL' with these items is shown in Figure 3-4. Please see *Inside Macintosh, Volume I*, page 421 for further information.

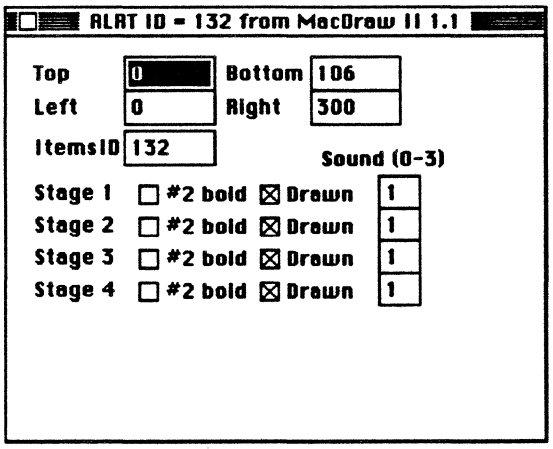
■ **Figure 3-3** Editing an 'ALRT' resource



■ **Figure 3-4** Special parameter strings



■ **Figure 3-5** 'ALRT' resource text view



'DITL' resources

For 'DITL' (dialog item list) resources, the editor displays an image of the items from the list as they would be displayed in a dialog or alert box. When you select an item, a size box appears in the lower-right corner of its enclosing rectangle so that you can change the size of the rectangle. You can move an item by dragging it with the mouse.

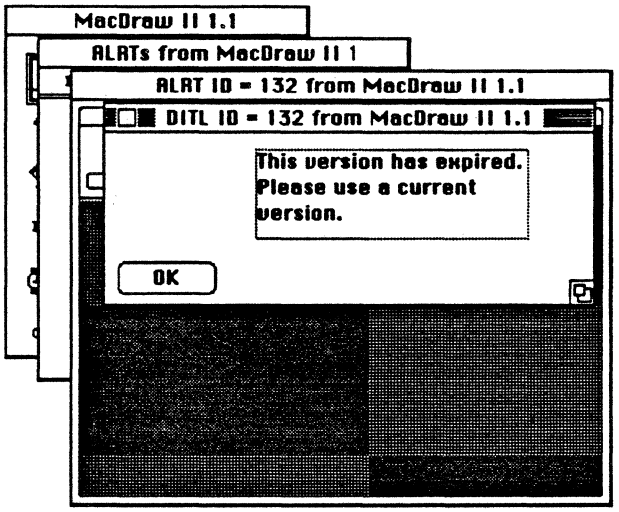
If you open an item within the dialog box, the editor associated with the item is invoked; for an 'ICON', for example, the icon editor is invoked. If you hold down the Option key while opening a 'CNTL', 'ICON', or 'PICT', the hexadecimal editor is invoked. If you hold down the Option and Command keys while opening a 'CNTL', 'ICON', or 'PICT' resource, the 'DITL' Item Editor (the editor used for buttons, static text, and so on) is invoked. Some dialog items are not editable and are listed as User Items. These are defined in the application, rather than in the Dialog Manager, and are actually built only when you run the application.

When you edit a 'CNTL' item, you will find that two rectangles are used to determine the location and size of the control. The location of the control within the 'DITL' is determined by the top and left values that you set in the 'DITL' Item Editor. The size of the control is determined by the size (bottom-to-top and right-to-left) that you set in the 'CNTL' editor. This means that no matter what you set the bottom and right values to in the 'DITL' Item Editor, they are reset to correspond to the size that is set in the 'CNTL' editor. You must edit both the 'DITL' item and the control itself to set both the location and size!

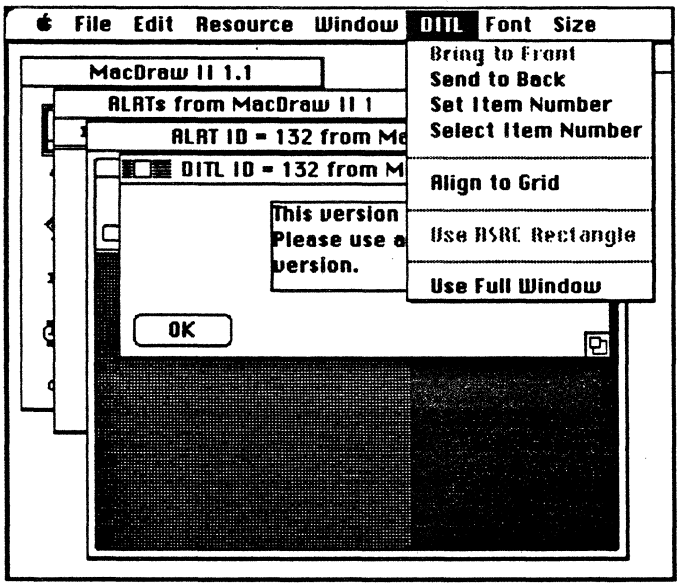
Because they are linked, the 'DITL' resource is usually given the same ID number as the parent 'DLOG' or 'ALRT'.

Figure 3-6 shows the 'DITL' corresponding to the 'ALRT' from Figure 3-4. The ALRT menu has been replaced by the DITL menu, shown in Figure 3-7.

■ **Figure 3-6** Editing a 'DITL' resource



■ **Figure 3-7** DITL menu



The DITL menu contains the following commands:

Bring to Front — Allows you to change the order of items in the item list. Bring to Front causes the selected item to be drawn in front of any items that it may overlap. The actual number of the item is shown by the 'DITL' Item Editor.

Send to Back — Causes the selected item to be drawn behind any items that it may overlap.

Set Item Number — Allows you to specify a new number for the selected item.

Select Item Number — Allows you to select an item by specifying its number.

Align to Grid — Aligns the item on an invisible 8-by-8-pixel grid. If you change the item location while Align to Grid is on, the location is adjusted such that the upper-left corner lies on the nearest grid point to the location you gave it. If you change the item size, it is constrained to be a multiple of 8 pixels in each dimension.

Use RSRC Rectangle — Restores the enclosing rectangle to the rectangle size stored in the underlying resource. Note that this command works on 'ICON', 'PICT', and 'CNTL' items only; the other items have no underlying resources.

Use Full Window — Adjusts the window size so that all items in the item list are visible in the window. The window size that your program will use when it displays the 'DITL' is actually stored in the parent 'ALRT' or 'DLOG' resource; this command is present solely for your convenience when you are editing the dialog items.

Font and Size menus are also present. These menus are provided to allow you to see how your 'DITL' looks when displayed in various typestyles. The font and size you set by using these menus are not saved, and must be reset each time you edit the 'DITL'.

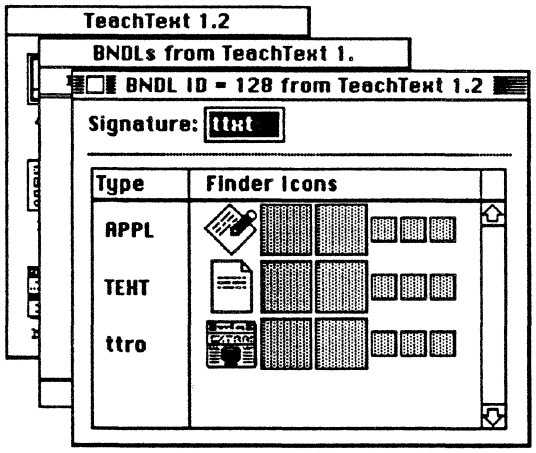
'BNDL' resources

To date, 'BNDL' resources have been mysterious, opaque, and difficult to learn about.

For historical reasons they have a fairly complex set of concepts behind them, but in fact, the only thing they do is bring together an application's documents (including the application file itself) and their icons for the Finder. Any application that has a distinct icon on the desktop also contains a 'BNDL' resource. For more details on the structure and concept of the 'BNDL' resource itself, please refer to Appendix C, "The 'BNDL' Resource."

The 'BNDL' editor in ResEdit 2.0 helps you create a bundle consisting of the necessary 'BNDL', 'FREF' and Finder icon resources, and saves you the burden of dealing with the internal workings of the bundle concept. The basic view you get when you first bring up the 'BNDL' editor is shown in Figure 3-8. The window appears in the display with the largest available number of gray levels or colors. (This is also true of the extended view, shown in Figure 3-10.)

■ **Figure 3-8** Editing the 'BNDL' resource, simple view

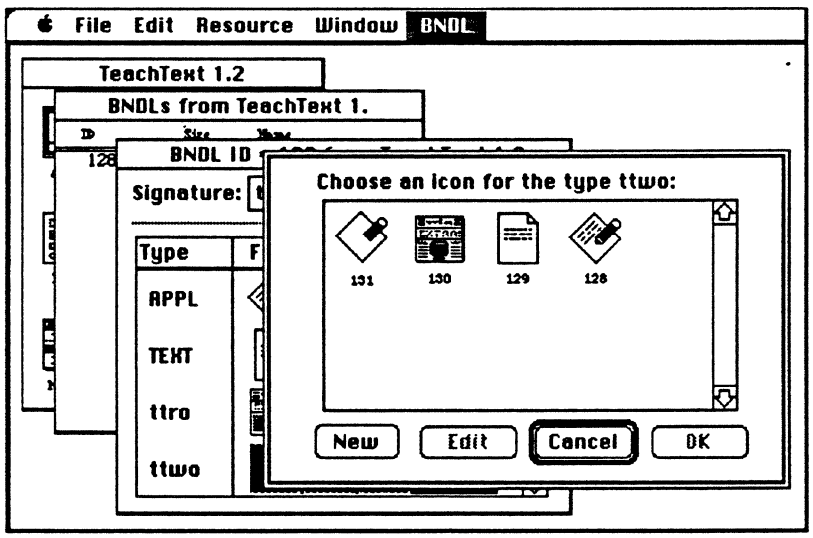


The Finder bundles together documents, applications and their icons with a 4 character signature, which must be unique for every application. All the necessary resources to do this are stored in the so-called Desktop file (or in the desktop database in system software version 7.0). This signature is shown in the first line of the window. All characters in the Macintosh character set (see Appendix D) are allowed in the signature. In order to register a unique signature for your own application, please contact Macintosh Developer Technical Support at Apple.

This signature is used as the creator code for all files that are part of the bundle (the creator code is a property of every file and can be set using the Get File/Folder Info command on the File menu). Every file on the Macintosh also has a file type, which is another 4 character field (several standard file types are defined: APPL for application, TEXT for plain text document, PICT for picture files, etc.). This file type is not only used to differentiate between different kinds of files but is also used to associate distinct icons with different files having the same creator (i.e. belonging to the same application). This is what the list in the bottom part of the 'BNDL' editor window does. In order to create a new file type and its icon, select Create New File Type from the Resource menu. Enter the file type in the left column and open the Finder Icon field in the right column by selecting Choose Icon from the BNDL menu or by double-clicking on the field.

Figure 3-9 shows the Icon chooser. Here you can either choose an existing icon for your file type, or you can create your own by pressing the "New" button. Note that even though the 'BNDL' editor shows the entire Finder icon family, because of screen real estate considerations you will only see a list of 'ICN#' resources in this window.

■ **Figure 3-9** The Icon chooser



Once you have associated all your file types with distinct icons (remember to include the file type APPL for your application itself) there are only a few more steps necessary in order to make the Finder display your icons.

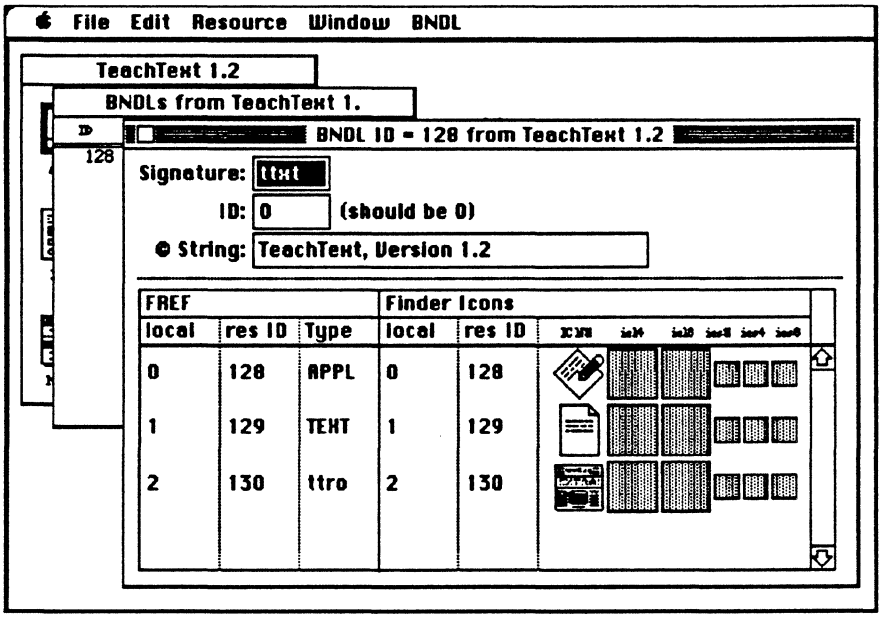
Select the Get File/Folder Info command from the File menu and choose your application in the upcoming list of files. Now set the file type to APPL and the creator to the signature you have entered in the 'BNDL' resource. Then set the Bundle bit and clear the Inited bit. This tells the Finder that your application contains a 'BNDL' resource and that it hasn't already seen your file. If the Finder doesn't immediately show your new icon, select your application and use the Get Info command in the Finder.

- *Note:* Once the Finder has seen your 'BNDL' resource and loaded the icons into its Desktop file, it will never again look at your 'BNDL', even if you clear the Inited bit. In order to change the 'BNDL' resource or to change some icons, you will need to remove your 'BNDL' resource from the Desktop file manually using ResEdit (this works, but is not recommended), or to recreate the Desktop file. To do this, hold down the Option and Command keys while restarting your Macintosh. The Finder will then ask you if you want to rebuild the Desktop file. Remember that when you do this, you lose all comments you may have entered in the Get Info windows in the Finder in system software previous to system software version 7.0.

If you want to move information contained in the 'BNDL' resource from one file to another you can do so by using the commands on the Edit menu. For copying operations, all necessary information (including the Finder Icons) is copied with the file type. If you clear or cut a file type in the 'BNDL' resource, please note that for safety reasons the Finder Icons are not removed (beautiful icons are so hard to design, it is generally considered better to waste a few bytes than accidentally kill one).

Should you ever have need to tinker with the internal workings of the 'BNDL' resource, you can edit all information stored in the 'BNDL' and associated 'FREF' resources by selecting Extended View from the BNDL menu. See Figure 3-10.

■ **Figure 3-10** Extended view in the 'BNDL' editor



For historical reasons the third line of the extended view, which displays the contents of the signature resource, is labeled "© string". This is because in the days before the introduction of the 'vers' resource to keep track of version information, the signature resource was used to store such information. Today the contents of the signature resource are ignored by the Finder unless the 'vers' resources are missing. In this case the Finder displays the contents in its Get Info window.

Editing 'cicn' resources

Ordinary color icons are pictorial resources of type 'cicn'. Figure 3-11 shows the 'cicn' editor. Because of the limitations of laserprinting, it is, unfortunately, not possible to do even a full grayscale version of this screen here. This version is far from accurate, and is intended only as a general representation. Please see the inside front cover for a color illustration of the 'cicn' editor.

Tools

Most of the tools in the tool palette are familiar from paint programs. One tool, the color-dropper, is new, and one, the pencil, has slightly different functionality when a color icon is being edited.

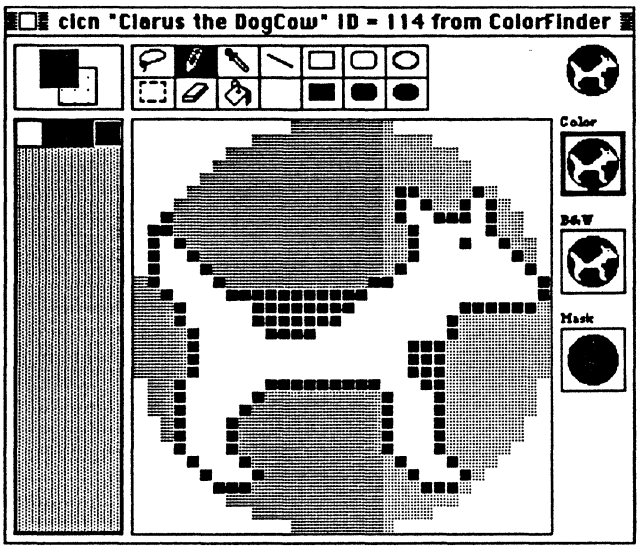
The color-dropper, when clicked on a pixel in the editing area of the window, sets the current color to be the color of that pixel. This is particularly useful when there are many available colors to choose from. The current color selection in the color part of the editor is independent from the current selection in the black and white part.

When you are using other drawing tools (e.g., the paint bucket), you can get the color-dropper by holding down the Option key. This does not, however, work with the eraser; the eraser always erases to white.

Clicking on any pixel that is not of the current color with the pencil tool changes it to the current color. Clicking on a pixel that is already the current color changes it to white.

It is possible to transfer images among the various framed images at the right edge of the 'cicn' editor. If you drag across either the color image or the black-and-white image, an outline will detach. You can then move that outline to the other image or to the mask. The destination highlights to indicate that releasing the mouse button will transfer the image. If you transfer the image to the mask, interior bits in the image are set to black.

■ **Figure 3-11** Color icon editing

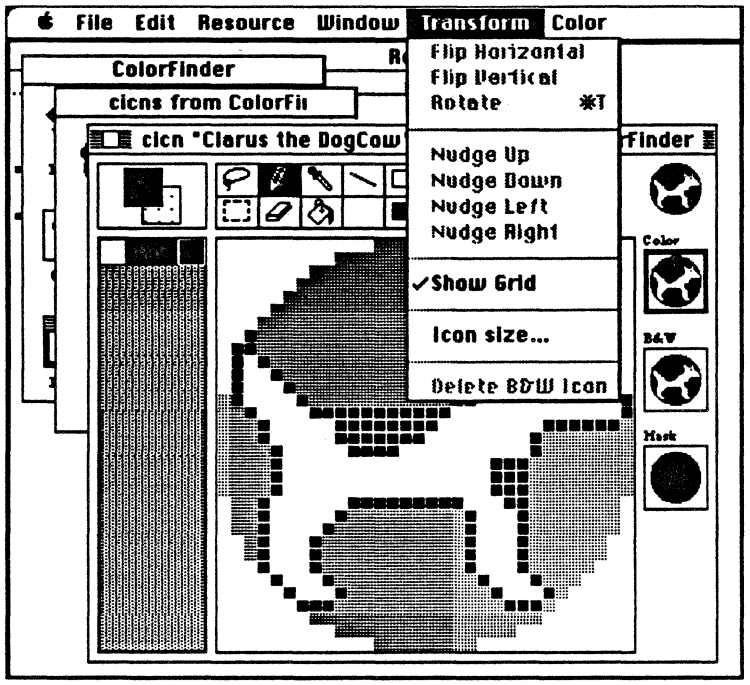


The Transform menu

The Transform menu is shown in Figure 3-12. It allows you to transform selected regions in several ways. The Flip Horizontal, Flip Vertical, and Rotate commands are familiar from paint programs. The Nudge commands move the selected region by 1 pixel in the indicated direction. (You can also nudge the selected region by using the Arrow keys.) The Show Grid command toggles. When turned on, it causes the icon to display with 1 screen-pixel of blank space around each enlarged pixel of the icon being edited. (When this command is turned off, the enlarged pixels of the icon touch each other. In Figures 3-11, 3-12, and 3-13, the Show Grid command is active, but because of the way the screen shots have been edited for printing, it is hard to see except in the black areas of the large view of Clarus.) The "Icon size" command brings up a dialog box that allows you to choose the horizontal and vertical sizes of the icon. These sizes are separate; that is, the icon does not have to be a square. The minimum for both is 8 pixels, and the maximum is 80. The Delete B&W Icon command is only active when the Black and White icon is selected.

It is possible to create a 'cicn' resource without a B&W image, but because the system uses the B&W image to display the icon on monitors that are set to black and white or to 4 grays or colors, it is probably a good idea to include it.

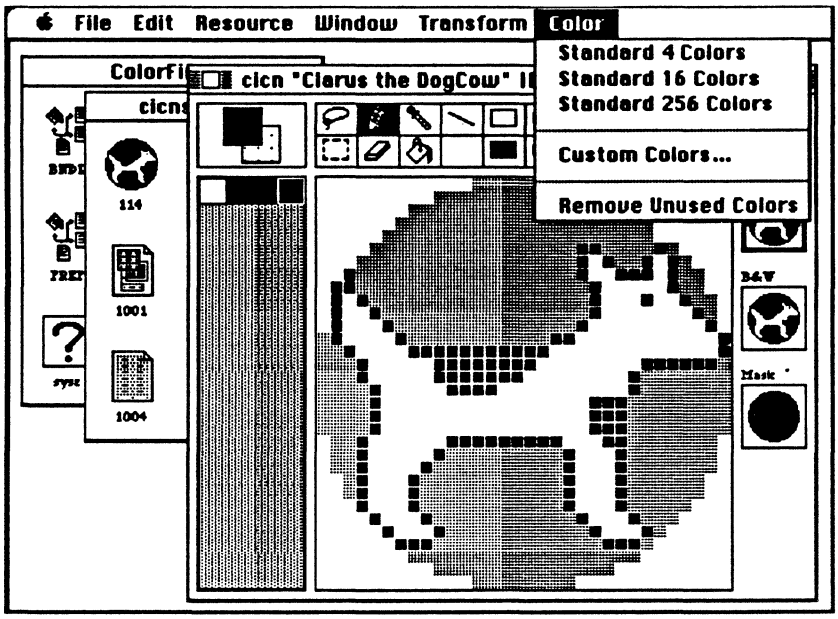
■ **Figure 3-12** Color icon editor with Transform menu



Creating new color icons

When you create a new 'cicn' resource, you get the default set of 16 colors. The color menu, shown in Figure 3-13, lets you select other color collections. The most commonly used collection is Standard 256 Colors, which installs the 8-bit System color table into the icon. Apple recommends that you use colors in the standard 16- and 256-color collections, as these are the colors that are typically present when a 'cicn' icon is drawn.

■ **Figure 3-13** The Color menu



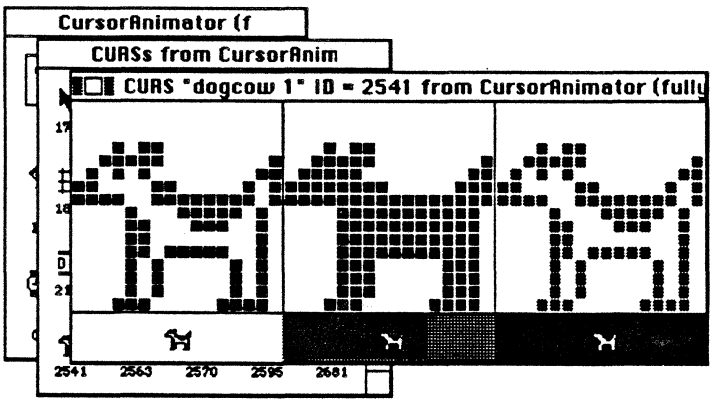
The Custom Colors command lets you use a previously saved 'clut' or 'pltt' resource instead of the standard 'clut' resources.

The Remove Unused Colors command is helpful at the end of editing, to minimize the size of the resource. If you use it before you are done, however, it eliminates all the colors you have not yet used, and if you later find you want some of them, you will have to add them back in.

'CURS' resources

Cursors are pictorial resources of type 'CURS'. Figure 3-14 shows the 'CURS' editor. The top part of the display has three large images for editing. The left image shows the cursor itself. The middle image is the mask for the cursor, which affects how the cursor appears on various backgrounds. The right image shows a gray picture of the cursor with a single point in black. This point is the cursor's "hot spot." (The hot spot is the point in the cursor that the Macintosh recognizes as the cursor's location. The hot spot of the familiar arrow cursor, for example, is its point.) You can invert bits in the left and center images by clicking them, and you can use the marquee tool to cut, copy, paste, and move part or all of the picture areas in the left and center images by holding the Shift key down and dragging, as with the other monochrome bit editors in ResEdit. In addition, if you click a pixel in the right image, that pixel becomes the cursor's hot spot. In the bottom part of the display, the cursor is drawn to scale on three different background patterns. To draw the cursor, a hole is made in the background by turning off the pixels in the area of the screen covered by the mask. Then the cursor is overlaid on the hole. Ordinarily, the mask should be just a filled-in outline of the cursor so that the cursor can be seen clearly.

■ **Figure 3-14** Editing a 'CURS' resource



The CURS menu contains the following commands:

Try Cursor	Lets you try out the cursor by having it become the cursor in use inside ResEdit.
Data -> Mask	Makes a filled-in copy of the cursor in the mask-editing area.

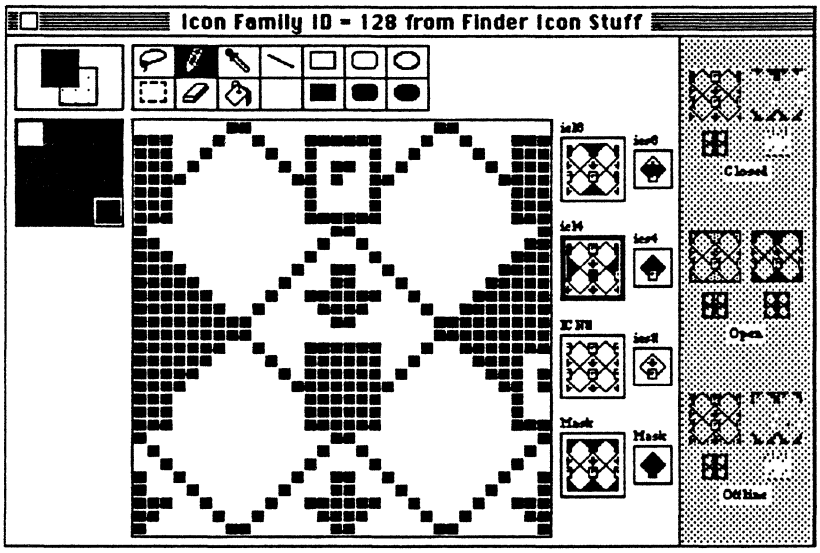
Finder icons

Finder icons, beginning with system software version 7.0, constitute a suite, or family, of pictorial resources. These include small and large color icons in 16 and 256 colors (types 'ics4' and 'ics8' in the small size, 'icl4' and 'icl8' in the larger size) as well as small and large monochrome icons, now types 'ics#' and the familiar 'ICN#', discussed later in this chapter. The large icons are 32-by-32 pixels, and effectively share the mask of the 'ICN#' type. The small icons are 16-by-16 pixels; they, too, share a common mask, in an 'ics#' resource.

Opening any of these resources except 'ICN#' automatically invokes the Finder icon editor and starts the subeditor for the particular resource type. 'ICN#' still has its own individual editor and may be edited either by itself or in the Finder icon editor with the other members of the suite. (Double-clicking a resource of type 'ICN#' opens the 'ICN#' editor rather than the Finder icon editor.)

Figure 3-15 shows the Finder icon editor during an 'icl4' edit. The other editing windows are quite similar, all of them sharing the tool palette; here, as with the 'cicn' editor, a monochrome illustration cannot fully represent the appearance of a color screen, but should give you some idea of the appearance of this editor. Please see the inside front cover for a color illustration of the Finder icon editor.

■ **Figure 3-15** Editing Finder icons



When you click one of the eight small pictures labeled with resource type names, that icon is opened for editing. Clicking in the display bar on the far right does nothing. This area shows the icon in the form of three groups of images against the selected background. The groups are labeled 'Closed', 'Open', and 'Offline'. The display shows the way the icons are drawn by the system software version 7.0 Finder. In each group, the icon is shown unselected on the left, and selected on the right.

Tools

Most of the tools in the tool palette are familiar from paint programs. One tool, the color-dropper, is new, and one, the pencil, has slightly different functionality when a color icon is being edited.

The color-dropper, when clicked on a pixel in the editing area of the window, sets the current color to be the color of that pixel. This capability is particularly useful when there are 256 colors to choose from. The current color is maintained independently in the black and white part of the editor, the 'icl4' and 'ics4' part, and the 'icl8' and 'ics8' part.

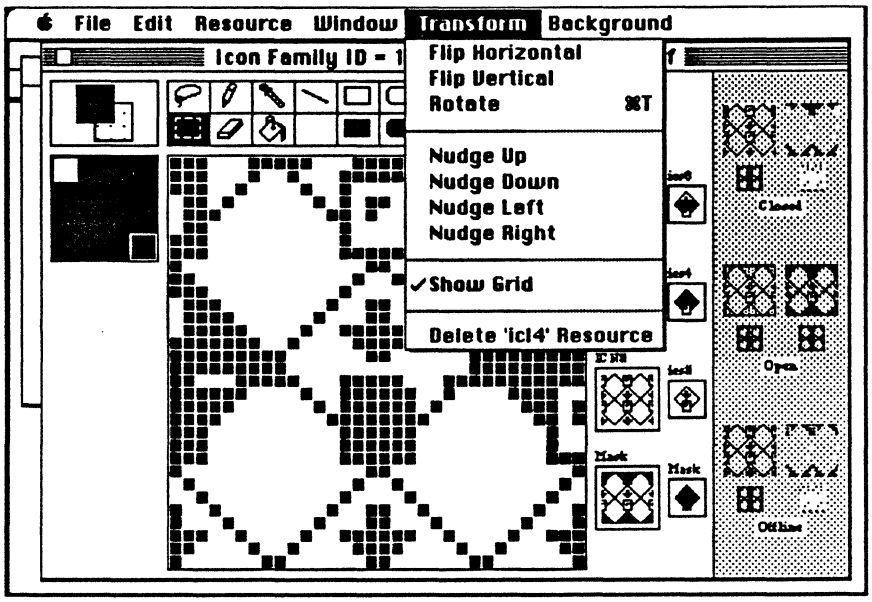
When you are using other tools (e.g., the oval-drawing tool), pressing the Option key activates the color-dropper. This is not true of the eraser, however.

When the current color is other than black or white, clicking on any pixel that is not of the current color with the pencil tool changes it to the current color. Clicking on a pixel that is already the current color changes it to white.

The Transform menu

The Transform menu is shown in Figure 3-16. It allows you to transform selected regions in several ways. The Flip Horizontal, Flip Vertical, and Rotate commands are familiar from paint programs. The Nudge commands move the selected region by 1 pixel in the indicated direction. The Show Grid command toggles and, when turned on, causes the icon to display with 1 screen-pixel of blank space around each enlarged pixel of the icon being edited. (When this command is turned off, the enlarged pixels of the icon touch each other. In Figures 3-10 and 3-11, the Show Grid command is active.) The Delete command allows you to delete the icon type currently being edited. If a mask is being edited, the Delete command allows you to delete the monochrome icon ('ICN#' or 'ics#') that owns the mask.

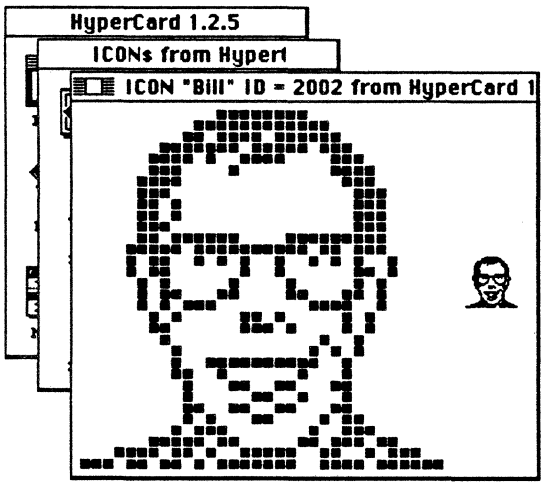
■ **Figure 3-16** Finder icon editor with Transform menu



'ICON' resources

When icons appear within a program (HyperCard is a good example), they are resources of type 'ICON'. The 'ICON' editor, as shown in Figure 3-17, displays one panel in the window. The left side of this panel shows an enlargement of the icon, and is an editing area. The right side of the panel shows the icon at actual scale. The editor for pictorial resources, including 'ICON', is a bit editor. It lets you click a pixel to invert it, and (if you hold down the Shift key) permits you to use the marquee tool to cut, copy, paste, and move part or all of the picture area. (Of course, you cannot move the entire picture.) If you cut or copy a marquee selection, you can paste it as a 'PICT' resource. First close the editor and picker. (You must close the picker in order for this to work.) If you then paste, ResEdit makes the contents of its scrap into a new 'PICT'. The 'PICT' resource picker does *not* have to be open when you cut, copy, or paste.

■ **Figure 3-17** Editing an 'ICON' resource



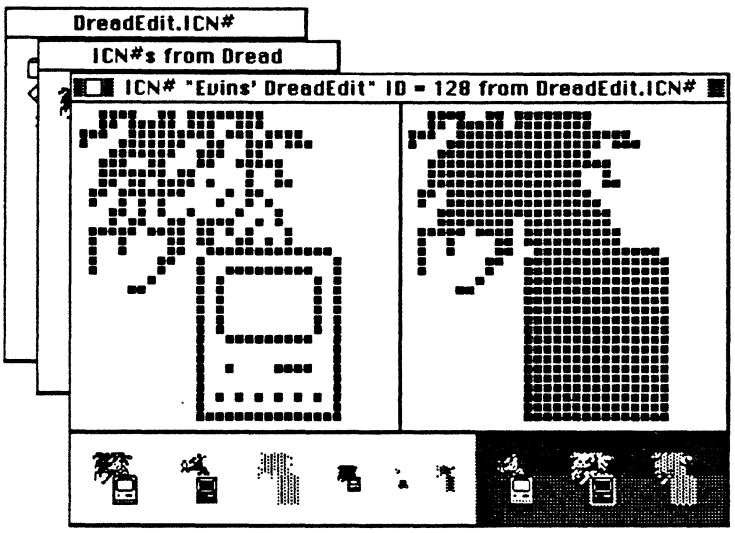
'ICN#' resources

The 'ICN#' resource is another common target for ResEdit. The icons that you see on the desktop in system software version 6.0 and earlier, representing applications and their documents, are all 'ICN#' icons, as are folder icons and even the trashcan. The 'ICN#' resource type is edited with a monochrome bit editor that permits you to change any of the pixels in the icon, which are in a 32-by-32-pixel square, and (if you hold down the Shift key) lets you use the marquee tool to cut, copy, paste, and move part or all of the picture, with the exception that if you use the marquee to select the entire picture and then you move it, you necessarily make part of it disappear outside the editing window. Arrow keys do *not* work here to move the selection. If you cut or copy a marquee selection, you can later paste it as a 'PICT' resource. See the description of 'ICON' resource editing earlier in this chapter.

In system software version 7.0 and later, this icon is part of the Finder icon suite, so it can also be edited with ResEdit's Finder icon editor. Because there is an 'ICN#' editor, however, that specific editor is activated if you double-click a resource of type 'ICN#'. You must bring up the Finder icon editor by opening another Finder icon type if you want to edit a resource of type 'ICN#' in concert with other Finder icons.

The 'ICN#' editor displays two panels in the window, as shown in Figure 3-18.

■ **Figure 3-18** Editing an 'ICN#' resource



The upper panel is used to edit the icon. It contains an enlargement of the icon on the left, and an enlargement of the icon's mask on the right. The lower panel shows, from left to right, how the icon will look unselected, selected, and open on both a white and a gray background. It also shows how the icon will appear unselected, selected, and open in the Finder small icon view.

In recent versions of the Finder, 'ICN#' resources are displayed on the screen as follows: First the mask is used to blank an area of the screen. Then an OR operation is performed in the same screen area, using the icon as data. (When a highlighted icon is displayed, the foreground and background "colors" (in this case black and white) are swapped before the OR operation is performed on the data.) If the mask is not the same shape as the outline of the icon, the results will in general be unaesthetic unless the background is black.

The ICN# menu contains the following commands:

Data -> Mask Makes a filled-in copy of the icon in the mask editing area.

Display using old method

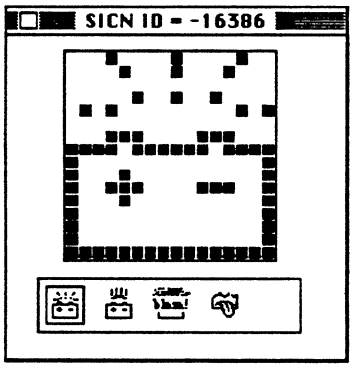
Lets you display the icon in the lower panel, using the method that was used by Finders in system software versions previous to version 6.0. If the mask is just a filled-in copy of the icon, you probably won't see a difference between the old and new displays.

'SICN' resources

Small icon ('SICN') resources are edited with a monochrome bit editor, just like other monochrome pictorial resources. Unlike 'ICON' or 'ICN#' resources, 'SICN' resources can, and usually do, occur in groups. A typical display is shown in Figure 3-19. The upper panel is enlarged and shows the icon currently being edited. The lower panel shows four icons at actual scale. The one shown in the upper panel is enclosed in a box in the lower panel. To get to a different icon, click its picture in the lower panel. If the icon you want to edit is not currently visible, click either the right or left picture, as appropriate, until it appears.

You can add a new icon before (to the left of) the currently selected icon by choosing the New command from the File menu. Commands on the Edit menu can be used to cut, copy, paste, clear, or duplicate icons.

■ **Figure 3-19** Editing a 'SICN' resource



'FONT' resources

The 'FONT' resource is one of two major ways of representing bitmap (screen) fonts for the Macintosh. (The 'NFNT' resource, described briefly later in this section, is the other.) The 'FONT' resource contains a series of pictures that typically represent items in the Macintosh character set, though they need not do so. A chart of the Macintosh character set is presented in Appendix D.

Because the Macintosh displays a character of type on its screen as a bitmap, however, it is possible for the pictures to be just that—pictures. 'FONT' resources on the Macintosh can contain scanned images and other pictures just as easily as they can contain the alphabet, numerals, and punctuation marks.

The Macintosh can modify elements of a font—for example, it can boldface them, or slant them for an approximation of italics. Print quality on dot-matrix printers (and screen-display accuracy as well) can be improved, however, by providing extra fonts that are constructed with those styles built into them. 'FONT' resources typically come in families, so that it is possible to display text on the screen (and print it on dot-matrix printers) in several styles, most commonly roman, bold, italic, and a bold-italic combination, without taking processor time to calculate the way such styles should look. These families can also correspond to downloadable PostScript fonts for laser printers and typesetters.

If you use ResEdit to examine a Fonts file from a recent Macintosh system software version, you will find that it contains three kinds of resources: 'FOND' 'FONT', and 'vers' (a record of the version number of the release). The 'FOND' resource "owns" one or more sizes of a particular font and contains kerning tables and other important information about the 'FONT' resources it owns. The 'FOND' resource has a unique ID number, from which the ID numbers of its subsidiary 'FONT's are calculated. To find the ID number of a particular 'FONT' resource, take the ID number of the parent 'FOND', multiply by 128, and add the point size of the 'FONT'. For example, 'FONT' ID 268 corresponds to New York (family ID 2), in 12 point size.

The ID numbers of 'FOND' resources may be from 0 (Chicago, the default System font) to 255, inclusive. Apple reserves ID numbers from 0 through 127. Unfortunately, there are a great many bitmap fonts (vastly more, in fact, than 255 of them), so occasional ID number collisions are unavoidable. Version 3.8 and later versions of the Font/DA Mover attempt to resolve such collisions, as do some third-party system-enhancer packages.

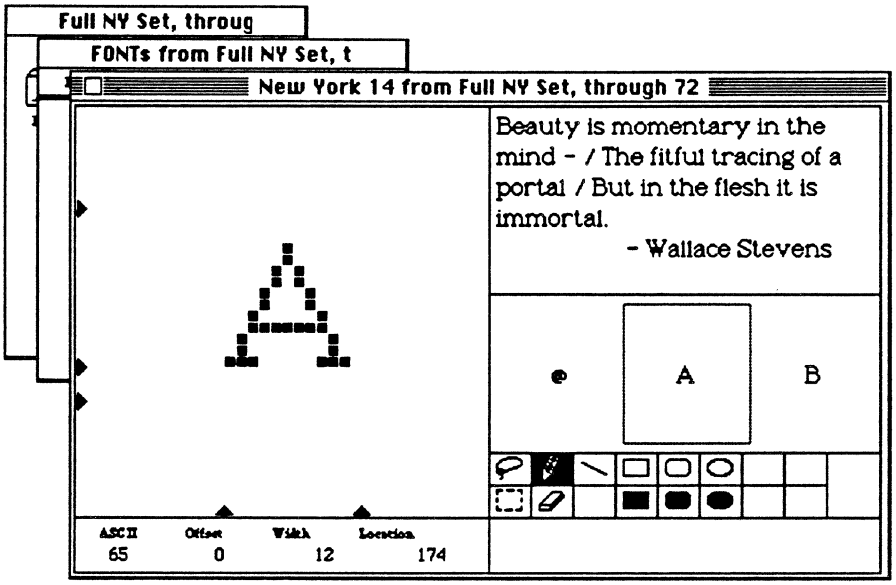
There is also another, newer kind of font resource, type 'NFNT'. Like 'FONT' resources, 'NFNT' resources are also owned by 'FOND' resources. ID numbering of 'NFNT' fonts is, however, not keyed to the ID number of the parent 'FOND'. Arbitrary numbering of 'NFNT' resources helps avoid font ID number collisions and facilitates resolution of conflicts when they do occur. 'NFNT' fonts, moreover, can contain and display more than 1 bit per pixel and can be assigned absolute colors with a corresponding 'fctb' resource, which is a ColorTable record. (Font ColorTable records are discussed in *Inside Macintosh,* Volume V, in the section on the Color Manager. The Font Manager is discussed in some detail in *Inside Macintosh,* Volumes IV and V.) ResEdit cannot edit 'NFNT' fonts, but it can copy and move them, as can version 3.8 and later versions of the Font/DA Mover. A third-party editor for 'NFNT' fonts is available.

Editing 'FONT' resources

Fonts are edited with a bit editor that is a superset of the bit editors for other pictorial resources. This editor has several of the tools you are probably familiar with from programs like MacPaint.

The editing window for 'FONT' resources is divided into four panels: a character-editing panel, a sample text panel, a character-selection panel, and a typical set of graphics tools. These panels are shown in Figure 3-20.

■ **Figure 3-20** Editing a 'FONT' resource



The *character-editing panel,* on the left side of the window, shows an enlargement of the selected character. You can edit it, as with the other bit editors for pictorial resources, by clicking bits on and off. Drag the black triangles at the bottom of the character-editing panel to set the left and right bounds of the character (that is, the character width). Two of the three triangles at the left side of the panel control the ascent and descent of characters in the font. If you want to increase the ascent or descent, move the appropriate triangle first. If you put pixels outside the indicated area and then move the triangle, those pixels are wiped out.

▲ **Warning** Changing the ascent or descent of a character changes the ascent or descent for the entire font. ▲

The third triangle on the left shows the location of the baseline, which is fixed and is displayed only for reference. Below the panel are the character number (labeled "ASCII"), and the character's offset, width, and location, all in decimal notation.

◆ *Note:* The correspondence between the Macintosh character set number and a real ASCII number is limited. Strictly speaking, ASCII is a set of 128 characters, numbered from 00 ($00, the NULL character) through 127 ($7F, the DEL character), and is intended to represent a basic character set rather than any font or typeface, in a relatively universally understood form. Because the Macintosh character set is oriented toward electronic publishing, which has more (and different) requirements, it has twice as many possible character numbers. (See the section on the 'KCHR' editor later in this chapter.) For ordinary text fonts, characters 0 through 127 of a Macintosh font are the ASCII character set. For Symbol, ITC Zapf Dingbats®, and the various pictorial fonts, however, the correspondence with the ASCII character set is minimal. The Macintosh character set is shown in Appendix D.

The *sample text panel,* at the upper right, displays a sample of text in the font currently being edited. (You can change this text by clicking in the text panel and using normal Macintosh editing techniques.)

The *character-selection panel* is below the text panel. You can select a character to edit by typing it (using the Shift and Option keys if necessary), or by clicking it in the row of three characters shown. To move upward through the character number range, click the right character in the row; to move downward, click the left character. The character you select is boxed in the center of the row. (To scroll quickly, click the right or left character and drag the pointer outside the selection panel, to the right or left.)

The *graphics tools panel,* directly below the character-selection panel, offers several familiar graphics-manipulation tools, including the pencil, eraser, circles, and rectangles. The 'FONT' editor, unlike the other bit editors, includes the marquee tool as a panel selection, and the lasso is also available.

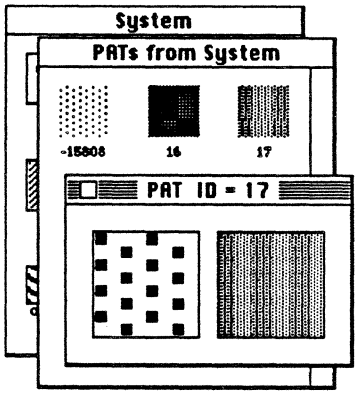
Any changes you make in the character-editing panel are reflected in the text panel and the character-selection panel, except on monitors displaying more than 2 colors or gray levels.

You can also change the name of a font. The font name is stored in two places: as the name of the 'FOND' resource of that font family, and as the name of the size 0 'FONT' resource. To change the font name, select the individual 'FOND' resource with the name you wish to change, and choose Get Info from the File menu. To maintain consistency, you should also change the name of the 0 point 'FONT' resource. This resource does not show up in the normal display of all fonts in a file. To display it, hold down the Option key while you open the 'FONT' type from the file window. You will see a generic list of fonts. Select the font with the name you wish to change, and choose Get Info.

'PAT ' resources

The 'PAT ' resource (pattern) editor is shown in Figure 3-21. It displays two panels, with the editing area on the left and the pattern shown on the right. The bit editor for 'PAT ' resources is very similar to the bit editor for other pictorial resources. It lets you invert a bit in the central editing area, and lets you use the marquee tool by holding down the Shift key while you drag. The editing area is small, but it is possible to make some use of the marquee tool.

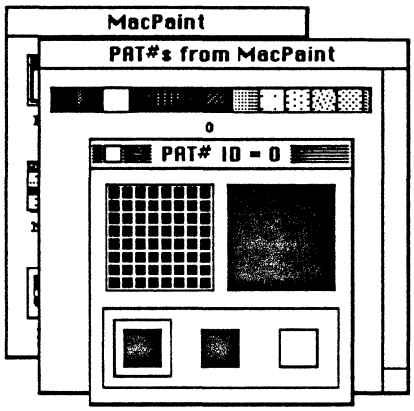
■ **Figure 3-21** Editing a 'PAT ' resource



'PAT#' resources

The 'PAT#' resource (pattern list) editor is a bit editor much like the 'SICN' editor; it is shown in Figure 3-22. Instead of displaying a single enlarged picture of the pattern being edited, it shows two. The one on the left is for editing; the one on the right shows the resulting pattern at full scale.

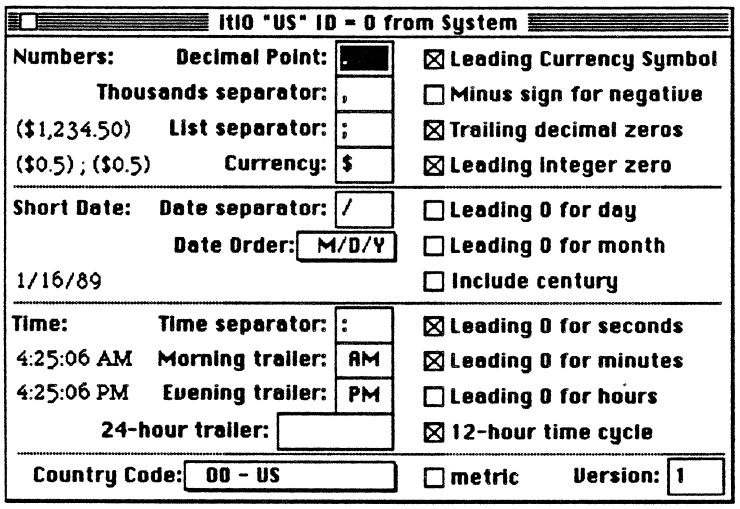
■ **Figure 3-22** Editing a 'PAT#' resource



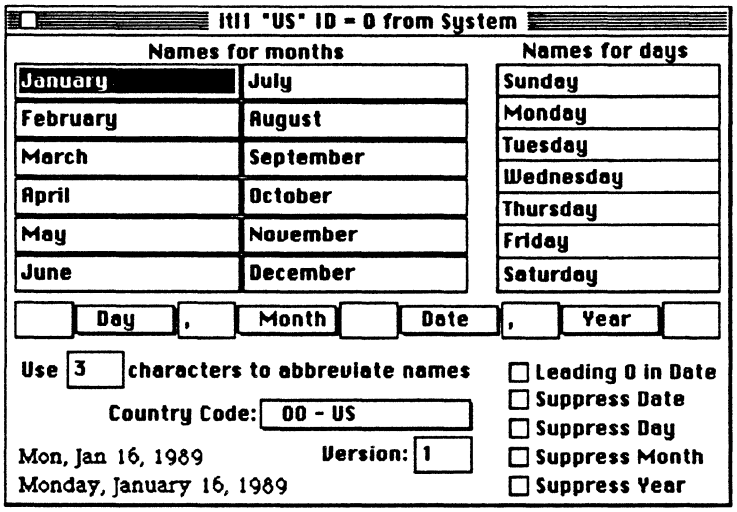
'INTL', 'itl0', and 'itl1' resources

The 'INTL' resource combines the functionality of the 'itl0' and 'itl1' resources. That is, 'INTL' "US" ID = 0 is the same as 'itl0' "US" ID = 0, and 'INTL' "US" ID = 1 is the same as 'itl1' "US" ID = 0. These resources are used in international localization. For further information, see *Inside Macintosh*, Volume V, Chapter 16. Each of these resources (whether you edit them as 'INTL' or as 'itl0' and 'itl1') is shown as a window with a set of boxes to be filled in and some buttons that can be clicked. Figures 3-23 and 3-24 show the windows for 'itl0' and 'itl1'.

■ **Figure 3-23** Editing an 'itl0' resource



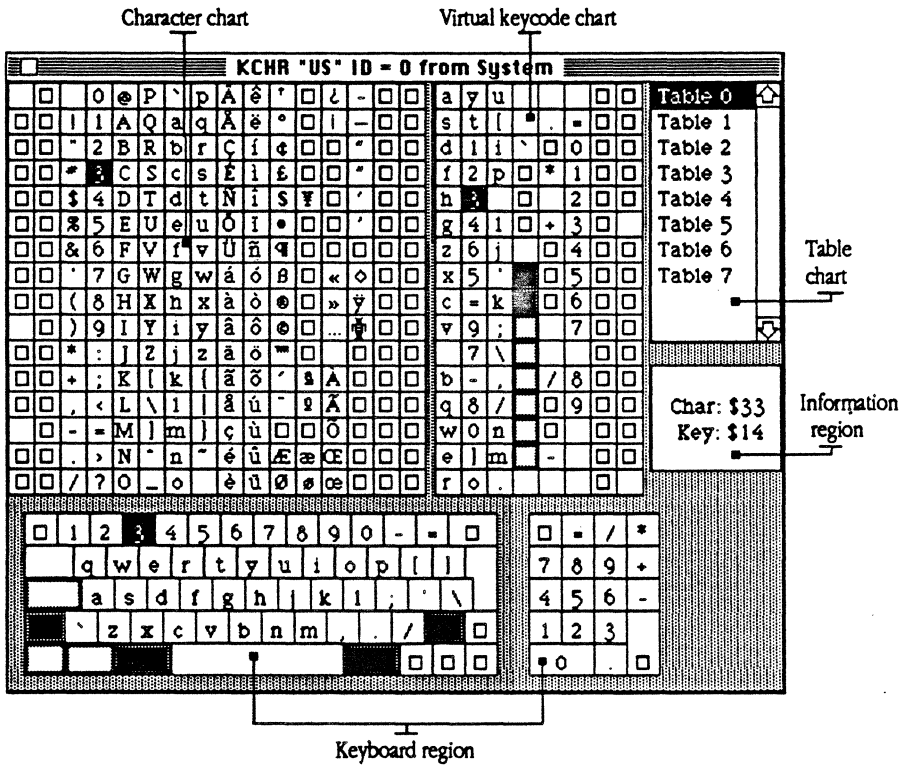
■ **Figure 3-24** Editing an 'itl1' resource



'KCHR' resources

The 'KCHR' resource controls keyboard mapping. The 'KCHR' editor can be used with any Macintosh that runs system software version 5.0 or later. The main 'KCHR' editing screen is shown in Figure 3-25, with Command-Option-3 pressed; the dead key editor is shown in Figure 3-26. Appendix A contains an in-depth discussion of the 'KCHR' resource itself, and a short section of 'KCHR' questions and answers appears in Chapter 6.

■ **Figure 3-25** Editing a 'KCHR' resource



The main 'KCHR' editor

The display for the main 'KCHR' editor (Figure 3-25) is divided into five parts, which are described in the sections that follow.

The character chart

This chart shows the 256 characters that make up the currently selected font. It displays the character generated by the currently pressed key, by highlighting it. You can also display a character by clicking with the mouse in either the keyboard region or the virtual keycode chart. These characters can be assigned to keys on the keyboard. To assign a character to a key, drag the character either to a keycap in the keyboard region or to the virtual keycode chart. You cannot assign characters to the Command, Option, Shift, Caps Lock, Control, Return, or Enter keys.

The table chart

The Shift, Caps Lock, Option, Command, and Control keys are considered to be "modifiers"; no combination of modifier keys generates a character code unless some other key is also pressed. The table chart shows which table is used by the currently depressed modifier key combination.

Please note that although there are 256 possible combinations of modifier keys, most versions of the 'KCHR' resource use only 8 tables, and very few ever use more than 16. This is because similar modifier key combinations are frequently mapped to the same table. For example, in the U.S. 'KCHR', all combinations involving the Control key point to Table 6. Also, the Caps Lock and Shift combination points to Table 1 (which is pointed to by the Shift key) rather than Table 2 (which is pointed to by the Caps Lock key on its own).

To change the table used by a modifier key combination, press that combination of modifier keys and click on a different table. The mapping is changed by the editor. This feature is probably of very little use, and the information is included here for completeness. Here is a listing of the tables as they are pointed to by various modifier key combinations in the U.S. 'KCHR', as supplied:

- Table 0 is shown with none of the modifier keys pressed, or with the Command key or Command and Shift keys pressed.
- Table 1 is shown with the Shift key or Caps Lock and Shift keys pressed.
- Table 2 is shown with the Caps Lock key pressed.
- Table 3 is shown with the Option key pressed.
- Table 4 is shown with Shift and Option keys pressed.
- Table 5 is shown with Caps Lock and Option keys pressed.
- Table 6 is shown with Option and Command keys pressed.
- Table 7 is shown with the Control key (and any other keys) pressed.

The virtual keycode chart

This chart shows all 128 keycodes in the current table, and highlights the keycode that is generated if you press a particular key with the current modifier key combination. These keycodes come from the keyboard, and are virtual in the sense that further translation has to take place before a Macintosh character set number results and a character can be displayed.

The keyboard region

This area reflects a particular keyboard layout. You can choose a different keyboard for displaying the virtual keycodes by using the "View as" command on the KCHR menu. The Apple® Extended Keyboard and Extended Keyboard II have two sets of modifier keys, and you can use the "Uncouple modifier keys" command, also on the KCHR menu, to get access to the alternate modifier keys (the ones on the right side of the keyboard, which are usually coupled with the ones on the left side). If you do not have the Apple Extended Keyboard or Extended Keyboard II connected to your Macintosh, you cannot choose the "Uncouple modifier keys" command.

Note that the modifier keys shown in the keyboard picture have a gray border. This border has two purposes:

- It reminds you that you cannot drag a character from the character chart onto a modifier key.
- It helps you find the modifier keys in the virtual keycode chart. (They have a gray border there, too.)

Note also that if you press the Option key, some keys in the display are shown with solid black borders. These are "dead" keys. If you click a dead key, the special editor for dead keys is invoked. For more information on editing dead keys, see "Editing Dead Keys," later in this chapter.

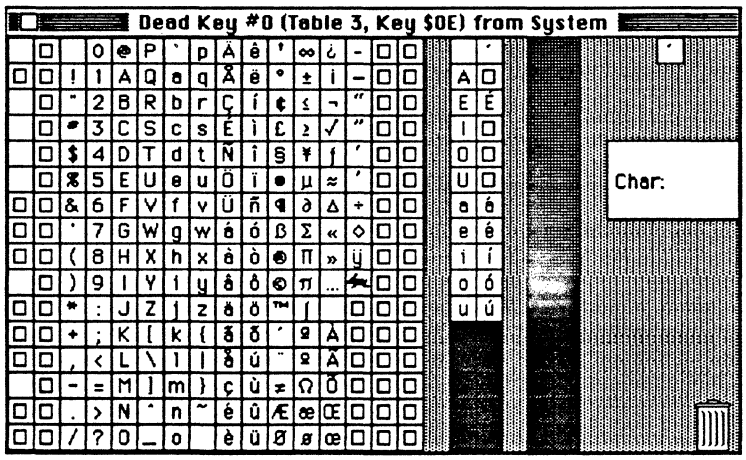
The information region

This small chart shows you the character code and virtual keycode, both in hexadecimal form.

Editing dead keys

Some combinations of keys do not immediately specify a character. Because nothing appears on the screen and the cursor does not move when these combinations are pressed, they are called "dead" keys. Typically they act to modify the next key that is pressed after the dead key is released. The special editor for dead keys is shown in Figure 3-26.

■ **Figure 3-26** Editing a dead key



The dead-key editor

The display for the dead-key editor is divided into five functional sections.

The character chart

This chart displays the character codes and is used to assign a different character code to either a completion character, a substitution character, or the nomatch character; you assign a code by dragging the character to its new location. If you drag a character to one of the empty slots (displayed in gray) in the completion and substitution character pair list, you automatically add a new pair.

The nomatch character

If the character typed after the dead key doesn't fit, a nomatch character is displayed, followed by the character you have typed. For example, Option-E must be followed by a vowel; it doesn't make much sense to put an accent mark on a *k*. The nomatch character for the current dead key is shown in the upper-right corner of the window.

The completion and substitution character pair list

This list shows the translation rules for the dead key that is currently selected. There are two columns, allowing for a total of 32 dead keys. The left half of each column shows all completion characters; the right half shows all substitution characters. If the character typed after the dead key is one of the completion characters, the matching substitution character is actually produced. For example, pressing Option-e and then e produces the character é.

The Trash

To remove a completion/substitution character pair, just drag either character from that pair in the completion/substitution pair list to the trashcan in the lower-right corner of the window.

The information region

This area contains the character code in hexadecimal form whenever you click one of the other parts of the editor. It is on the right edge of the window, and contains the word "Char:".

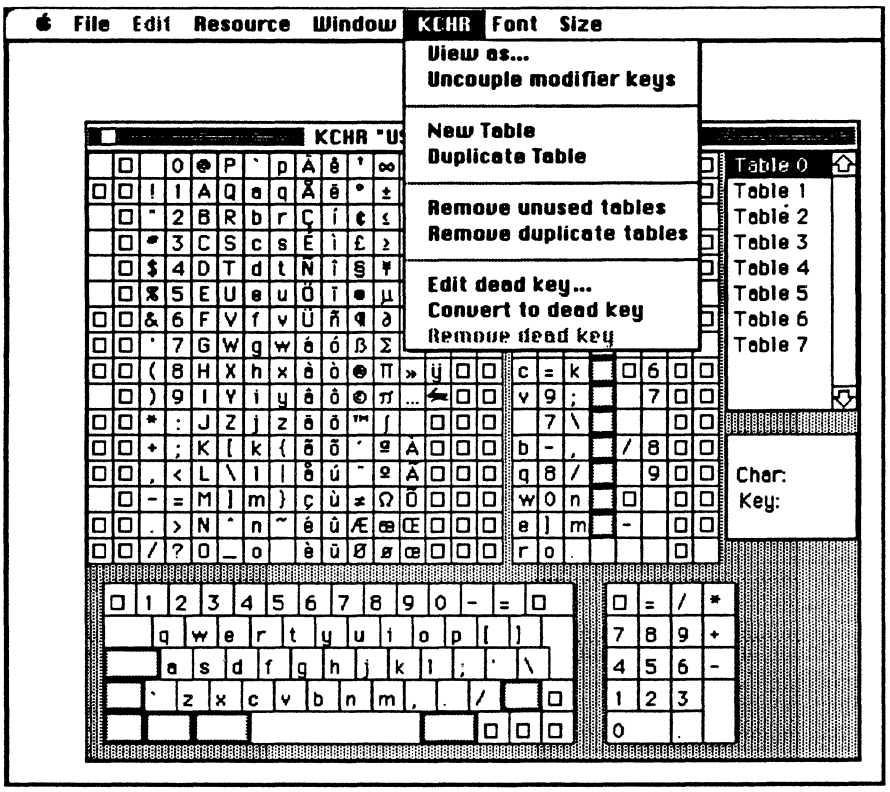
The menus

The 'KCHR' editor has three menus: KCHR, Font, and Size.

The KCHR menu

This menu is shown in Figure 3-27.

■ **Figure 3-27** The KCHR menu



The KCHR menu contains the following commands.

View as... If you have the Key Layout file (which has been part of the system software since version 4.2) in your System Folder, you'll be presented with a list of keyboards to be used for displaying the virtual keycodes. Note that you are *not* changing the layout of a particular keyboard, but the 'KCHR' resource that is used by *all* keyboards and is based on the ISO (International Standards Organization) ADB keyboard.

Uncouple modifier keys

This command is enabled when you have an ADB extended keyboard connected to your computer. It can be used to uncouple the right modifier keys (see note above) and thus edit the tables used by them. Please note that the 'KCHR' editor automatically recouples them whenever you bring another window to the front or close the editor.

- *Note:* When you select the "Uncouple modifier keys" command, you must also use the "View as" command to set the current keyboard to a keyboard that supports uncoupled modifier keys. To avoid confusion, and because not all keyboards support this decoupling, it is recommended that you not make use of this command.

New Table — Creates a new empty table.

Duplicate Table — Creates an identical copy of the current table.

Remove unused tables
Looks for tables that are not used by any modifer key combination, and removes them.

Remove duplicate tables
Checks for tables that are identical, reassigns modifier key combinations as necessary to one table, and removes the duplicate(s).

Edit dead key... — Displays a dialog box containing a list of all dead keys and lets you choose one to edit. Note that there is a shortcut to edit dead keys: You can either click a dead key on the screen, or press the dead key on the keyboard. In either case the dead-key editor will automatically pop up.

Convert to dead key
Whenever you hold down a key with any combination of modifier keys and choose this menu command, the key will be converted to a dead key. You can then use the Edit dead key command to define all valid completion and substitution characters for the new dead key.

Remove dead key — This command is enabled only when a dead-key window is open. It removes the dead key currently being edited from the dead-key list, converting it into a live key in the process.

The Font menu

This menu lets you choose a font for displaying the characters in the editor's window.

The Size menu

This menu lets you choose a size for the characters displayed in the editor's window. All characters in the window are automatically resized.

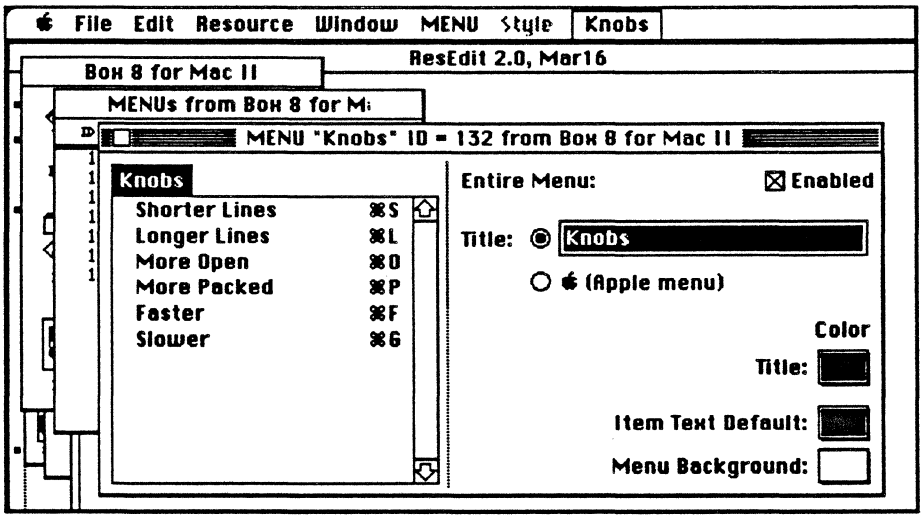
- *Note:* If you are editing 'KCHR' resources on a Macintosh SE, Macintosh Plus, or Macintosh 512K enhanced, the 'KCHR' editor automatically sets the size to 9 points so that the editing window fits on the screen.

'MENU' resources

Menus are an important part of the Macintosh user interface and are found in all applications and many desk accessories. They are stored in resources of types 'MENU' (regular menus), 'cmnu' (MacApp® temporary menus; these are converted into 'MENU' resources by PostRez during the MacApp build process, so you will never find one in an application), 'CMNU' (MacApp permanent menus; these will be supported in future versions of MacApp), and 'mctb' (menu color tables for any of the preceding types). The 'cmnu' and 'CMNU' types differ from regular menus in that they have an additional command number field stored for each item in the menu. ResEdit 2.0 supports editing of all these menu resource types with a new editor that automatically integrates the color information stored in the 'mctb' resources and thereby allows editing of menus in color. See the inside front cover for a color illustration of menu editing.

The display of the menu editor, shown in Figure 3-28, is divided into two sections. The left side shows the entire menu, and the right side displays detailed information about the item selected on the left side. To accommodate menus with many items, the box on the left side has a scroll bar.

■ **Figure 3-28** Editing a 'MENU' resource

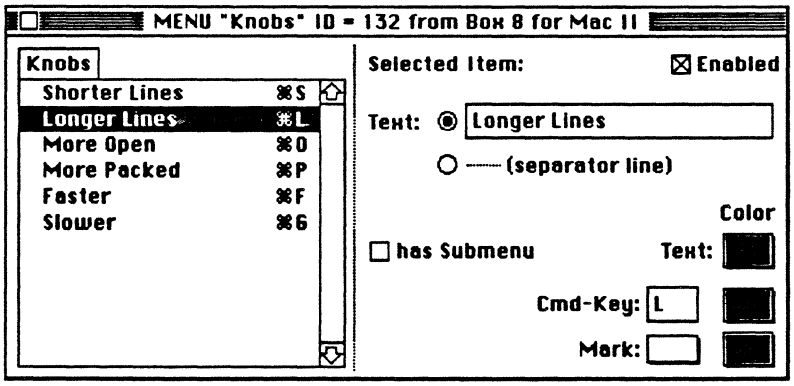


If the title of the menu is selected, the editor not only allows you to change the title but also displays some information about the entire menu. You can enable/disable the entire menu and also select colors for the menu's title, for the item text default, and for the menu background. On machines capable of displaying color the color patches pop up like menus and let you select a color from a palette corresponding to the pixel-depths of the deepest device intersecting the window. Should you, however, need to enter a color in RGB values, you can double-click on the color patches and set the color using the standard color picker. On monochrome machines the color picker is opened whenever you click the color patch, because a palette cannot be displayed adequately. Since the "Apple" character can't easily be generated on some keyboards there is also a convenient radio button to make the menu title the "Apple" character instead of text entered in the box. If you do enter the "Apple" character, the editor automatically chooses the radio button.

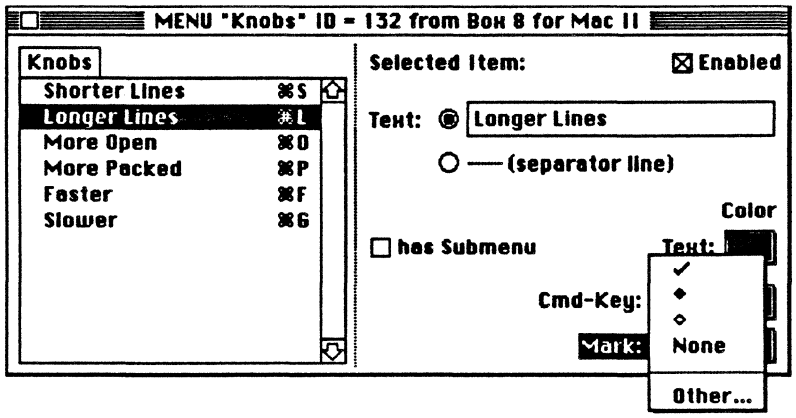
When you create a new menu, there are no items to select in order to start the editing process. You can choose Create New Item from the Resource menu, or type Command-K.

When an individual menu item is selected the display changes to the one shown in Figure 3-29. As in the title's display you can either edit the text of the item directly or you can use the radio button to make the item a separation line (which you can also do by entering "-" in the text box). You can use the Style menu to select a different style (bold, italic, and so on) for each item, and you can enable or disable the item with the checkbox in the upper right corner. For each item you can assign a command key equivalent (the menu manager is not case sensitive, so for esthetic reasons and consistency you should only use uppercase characters) and an item mark, which you can choose from an extensible pop-up menu shown in Figure 3-30. Both the command key equivalent and the mark character can be displayed in color. If you want to do that, select a color from the corresponding color palette pop-up menus.

■ **Figure 3-29** 'MENU' line item edit

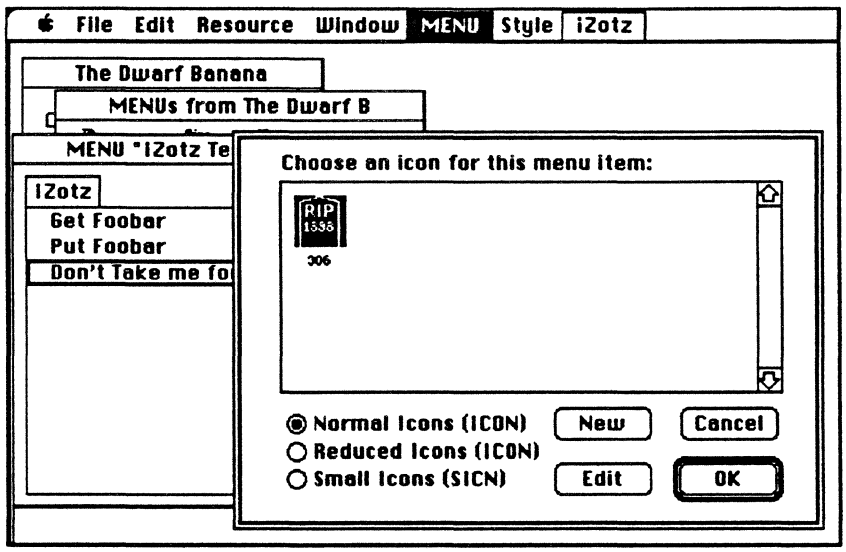


■ **Figure 3-30** 'MENU' mark pop-up



In order to make an important item look unique you can put an icon in front of the item's text. Select Choose Icon from the MENU menu to get the dialog shown in figure 3-31.

■ **Figure 3-31** 'MENU' Icon chooser



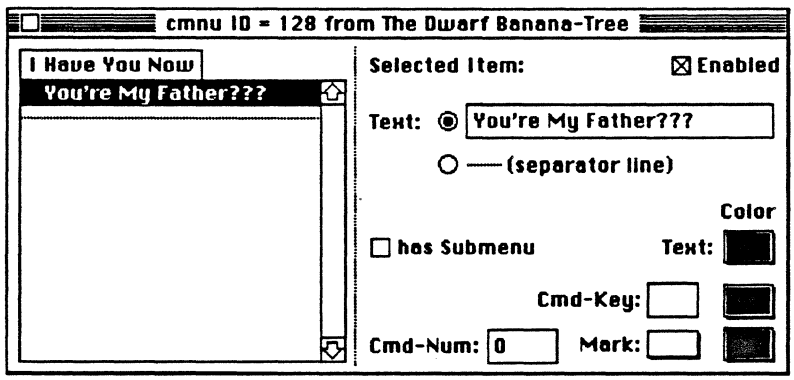
Because of menu manager restrictions, the icon's ID must be in the range of 257 to 511 in order for it to be used in a menu. All other icons are displayed in gray. If a regular item seems to be too large for your menu, you can select the "Reduced" radio button to shrink the icon to a more convenient 16x16 size or you can add a small icon (resource type 'SICN') instead of a regular one. If you later want to remove the icon from an item, choose Remove Icon from the MENU menu. In order to reduce clutter, the menu on the left side of the editing window does not show icons.

If you want to see how your menu looks in real life you can try it out at the right edge of the menu bar. To show you that this is not a regular menu but a sample of the menu you are editing, its title is outlined with a black border.

Sometimes a menu may become overcrowded with items. That's when you should start to think about organizing the items in groups and turning the menu into a hierarchical menu. The menu editor helps you create submenus by providing you with the option to turn any item into a submenu just by clicking in a checkbox. In order to edit the items of the submenu, either select Open Submenu from the Resource menu or double-click on the item's text.

If you happen to edit a 'cmnu' or 'CMNU' menu for inclusion in a MacApp program, you will notice that there is an additional field shown in the item's display that lets you set the command number for each item. This is shown in Figure 3-32, bottom center.

■ **Figure 3-32** 'cmnu' editing



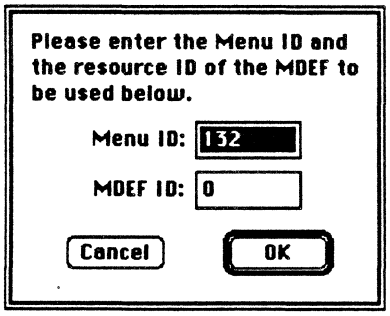
The menu editor also lets you rearrange the items in your menu. You can either use the standard commands on the edit menu, or you can put an item in a new position by dragging it around in the menu on the left side of the window. As you move the item around, a black line between items shows you where the item is currently located.

Selecting colors from the various pop-up palettes actually modifies an 'mctb' resource (menu color table) which is transparently generated and changed for you. If you want to get rid of the colors you have set, you can reset the 'mctb' resource by selecting Use Default Colors from the MENU menu.

The 'MENU' resource has two assigned ID numbers. One of these is the resource ID number; it is set by getting information on the resource from the picker window, and is the ID number that always shows up in the picker window. The other is the menu ID number; it is set inside the editor and is the part of the 'MENU' resource that is returned by the menu manager of the Macintosh Toolbox in response to MenuSelect and MenuKey calls. Keeping these two numbers the same, while not required, avoids confusion, and in fact they default to the same number. See Chapter 6 for more information.

The corresponding 'MDEF' ID number is almost always 0. This refers to the standard 'MDEF' in the System File, which is generally appropriate. Some menus do, however, need to be drawn differently. (Palettes, for example.) These could use separate 'MDEF' resources, and hence would not have 0 in this field. Figure 3-33 shows the 'MENU' and 'MDEF' ID number dialog box.

■ **Figure 3-33** 'MENU' ID dialog



Chapter 4 **Using ResEdit Templates**

One generic way of editing a resource is to fill in the fields of a dialog box. The contents of the dialog box are specified by a template contained, typically, in ResEdit's own resource file. This chapter discusses template editing.

Template characteristics

If you open an actual resource of any of the types listed in this chapter, you will find yourself editing in a dialog box, the contents of which are specified by the template of the same name as that resource type. (For example, the 'LAYO' resource, discussed further in Chapter 6, is controlled by the 'TMPL' resource named LAYO in ResEdit.) The template specifies the format of the resource and also specifies what labels should be put beside the editText items in the dialog box used for editing the resource.

- *Note:* Templates can contain a maximum of 2048 fields. For the purpose of enumerating, a field is defined as any item that is drawn on the screen. That is, a label counts as a field, as does a separator, and so on. This limiting number of 2048 is reached rather easily, particularly in resources with repeating lists, as for example, 'pltt'.

The 'TMPL' resource inside ResEdit is recursive, in the sense that the contents of each of these named 'TMPL' resources is a template for a template. (There is even, of course, one for 'TMPL' itself.) As of mid-1990, ResEdit contains 'TMPL' resources for these resource types:

'actb'	'acur'	'ALRT'	'APPL'	'BNDL'	'cctb'
'clut'	'cmnu'	'CNTL'	'CTY#'	'dctb'	'DITL'
'DLOG'	'DRVR'	'FBTN'	'fctb'	'FDIR'	'finf'
'fld#'	'FOND'	'FONT'	'FREF'	'FRSV'	'FWID'
'icmt'	'inbb'	'indm'	'infa'	'infs'	'inpk'
'inra'	'insc'	'itlb'	'itlc'	'itlk'	'LAYO'
'MBAR'	'mcky'	'mctb'	'MENU'	'nrct'	'PAPA'
'PICT'	'pltt'	'POST'	'ppat'	'PRC0'	'PRC3'
'PSAP'	'qrsc'	'RMAP'	'ROv#'	'scrn'	'SIGN'
'SIZE'	'STR '	'STR#'	'TEXT'	'TMPL'	'vers'
'wctb'	'WIND'	'wstr'			

Editing

When you are editing a template, the Tab key moves you forward from field to field within the template. Shift-Tab moves you backward. Here, however, the term *field* means an active area with an editable value in it. Fields are shown on the screen as boxes.

To add a new field to a repeating sequence in a template, select a separator, which is usually a set of asterisks (*****), and choose Create New Field from the Resource menu.

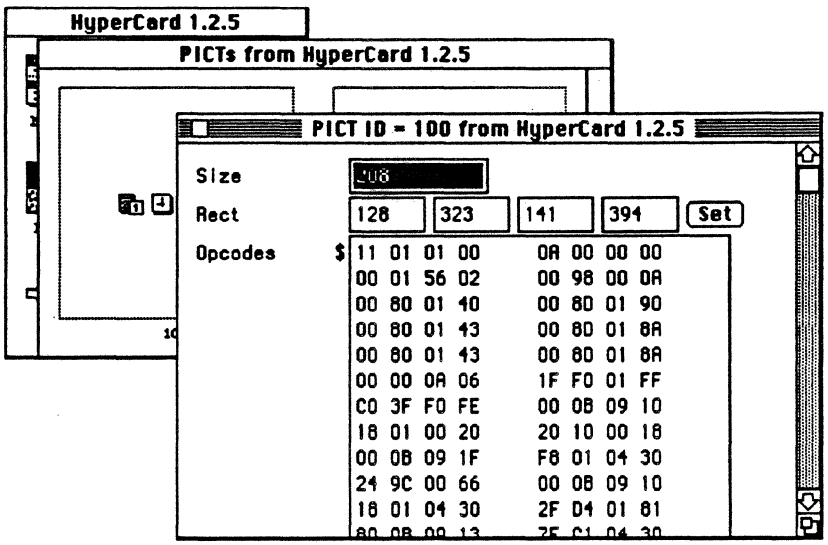
Some templates control windows or resources that contain rectangles. Some of these templates will have a Set button that lets you draw a rectangle on the screen to delimit the resource. The pixel numbers for the rectangle are automatically copied to the appropriate fields in the template. There is a Set button in the 'LAYO' template, which is discussed in Chapter 6; another is shown in Figure 4-1.

Values can be entered into numeric fields in either decimal or hexadecimal notation. You can enter a hexadecimal number into any numeric field by preceding it with a dollar sign ($).

'PICT' editing

There is no custom editor for 'PICT' resources, though there is a custom picker. 'PICT' resources can, however, be sized with the template that exists for them, which is shown in Figure 4-1. If you click the Set button, you can then draw a rectangle on the screen to define the shape and size of the picture. Otherwise, you can enter values in the fields as you would in any template.

■ **Figure 4-1** The template editor for 'PICT'



For other examples of template editing, see the description of the 'STR#' resource template in Chapter 5 and the description of the 'LAYO' resource in Chapter 6. Procedures for generating new templates are also covered in Chapter 5.

Chapter 5 **Creating ResEdit Templates**

This chapter describes how you can generate templates for your own resource types. These templates, which are resources of type 'TMPL', need not reside within ResEdit. The ResEdit Preferences file in the System folder is a good place to keep them.

Template example

The 'TMPL' resource inside ResEdit with name STR# is shown in Figure 5-1. It is shown here as a ready example of what 'TMPL' innards look like on the screen.

■ **Figure 5-1** 'TMPL' definition for type 'STR#'

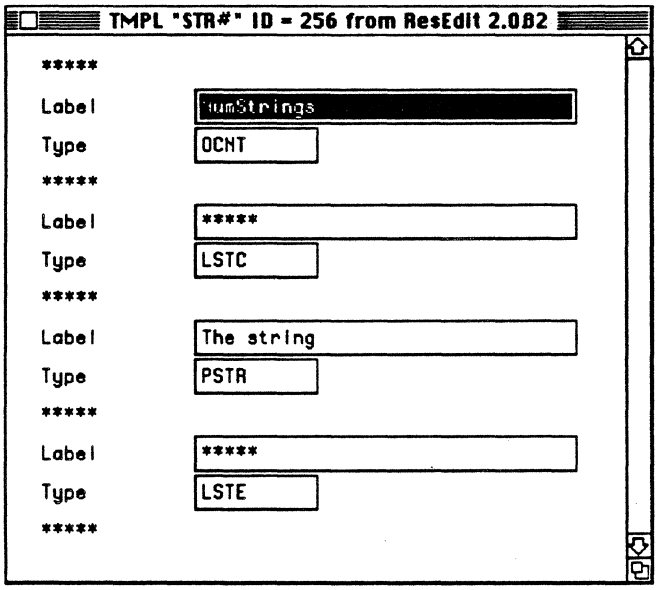
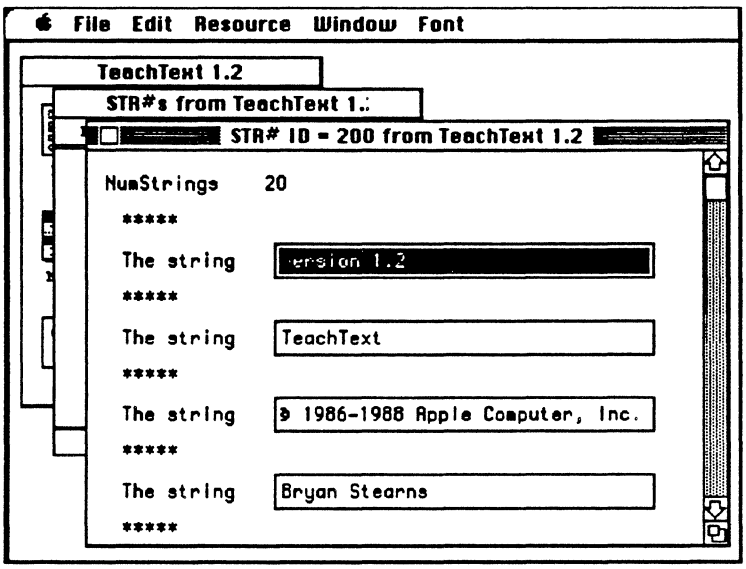


Figure 5-2 shows the same template being used to edit an actual 'STR#' resource. You can see the correspondence between the items in the 'TMPL' resource and the resulting display.

■ **Figure 5-2** 'STR#' template in use



You can look through the other templates and compare them with the structures of their corresponding resources to get a feel for how you might define your own resource template. (If you use MPW, note that these templates are equivalent to the resource type declarations contained in the {RIncludes} directory—refer also to the DeRez command in the *MPW Reference,* and the appropriate chapters of *Inside Macintosh.*)

These are the types you may choose from for your editable data fields:

DBYT, DWRD, DLNG	Decimal byte, decimal word, decimal long word.
HBYT, HWRD, HLNG	Hex byte, hex word, hex long word.
AWRD, ALNG	Word align, long align.
FBYT, FWRD, FLNG	Byte fill, word fill, long fill (with 0).
HEXD	Hex dump of remaining bytes in resource. (This can only be the last type in a resource.)
PSTR	Pascal string (length byte followed by the characters).
LSTR	Long string (length long followed by the characters).

WSTR	Same as LSTR, but a word rather than a long word.
ESTR, OSTR	Pascal string padded to even or odd length (needed for DITL resources).
CSTR	C string (characters followed by a null).
ECST, OCST	Even-padded C string, or odd-padded C string (padded with nulls).
BOOL	Boolean (two bytes).
BBIT	Binary bit. (There must be 8 or an even multiple of 8 of these; if fewer than 8 bits are defined, you must include placeholder bits.)
TNAM	Type name (four characters, like OSType and ResType).
CHAR	A single character.
RECT	An 8-byte rectangle.
H*nnn*	A 3-digit hex number; displays *nnn* bytes in hex format.
C*nnn*	A C string that is *nnn* bytes long. The last byte is always a 0, so the string itself occupies the first *nnn-1* bytes.
P*0nn*	A Pascal string that is *nn* bytes long. The length byte is not included in *nn*, so the string occupies the entire specified length.

- *Note:* Scrolling can become extremely slow if a template contains many BBIT or BOOL items.

ResEdit does the appropriate type checking for you when you put the editing dialog window away.

The template mechanism is flexible enough to describe a repeating sequence of items within a resource, as in 'STR#', 'DITL', and 'MENU' resources. You can also have repeating sequences within repeating sequences, as in 'BNDL' resources. To terminate a repeating sequence, put the appropriate code in the template as follows.

LSTZ	
LSTE	*List Zero-List End.* Terminated by a 0 byte (as in 'MENU' resources).
ZCNT	

LSTC

LSTE — *Zero Count/List Count–List End.* Terminated by a zero-based word count that starts the sequence (as in 'DITL' resources).

OCNT

LSTC

LSTE — *One Count/List Count–List End.* Terminated by a one-based word count that starts the sequence (as in 'STR#' resources).

LSTB

LSTE — Ends at the end of the resource. (As in 'acur' and 'APPL' resources.)

The "list-begin" code begins the repeating sequence of items, and the LSTE code is the end. Labels for these codes are usually set to the string "******". Both of these codes are required. It is generally advisable to keep the beginning and ending labels identical to each other, and to have them be no more than five characters long.

Your template does not have to be inside ResEdit; it can be in any open file. (The preferred location is the ResEdit Preferences file in your System Folder.) Note that if more than one currently open file contains a template for your resource type, the one in the most recently opened file is used when you edit resources of your type. To create a template, follow these steps:

1. Open the file that you want to put your template into.
2. Open the 'TMPL' type window. Use the Create New Resource command to create the 'TMPL' type if it doesn't already exist in the file.
3. Choose Create New Resource from the Resource menu.
4. Select the list separator (*****) by clicking it.
5. Choose Insert New Field(s) from the Resource menu. You may now begin entering the *label,type* pairs that define the template. Before closing the template editing window, choose Get Info from the Resource menu and set the name of the template to the four-character name of your resource type.
6. Close the file window and save changes.

The next time you try to edit or create a resource of the new type, you'll get the dialog box in the format you have specified.

Chapter 6 **ResEdit Tips**

As with any other utility, ResEdit takes some getting used to. This chapter presents a few handy tips and a few "hints and kinks" to help you become more comfortable with the capabilities of the program.

Hints and kinks

- At the risk of being slightly repetitive, and because these things can be important, it is once again suggested that you edit resources in a copy of your target file, rather than the original.
- If you choose Get Info for ResEdit (from the Finder), you will find that Application Memory Size is set to 500 KB. If you are editing large resources 500 KB is not sufficient, and you should give ResEdit more memory.
- The following sequence of steps can be used to copy a 'PICT' resource from most drawing or painting programs into another file:
  1. Open the file that contains the graphic that you want to turn into a 'PICT'.
  2. Select and copy the part of the graphic that you want.
  3. Start ResEdit and open the file that you want to contain the 'PICT' resource.
  4. Open the 'PICT' picker for that file.
  5. Choose Paste.

If you paste with the file window open instead of the 'PICT' picker window, you will get both the 'PICT' and the application's private resource type (for example, 'MDPL' if your 'PICT' is from MacDraw).

- To add a picture to a 'DLOG':
  1. Get a picture. Add it to the 'PICT' resources in your file. (See the previous tip.)
  2. Choose the Get Resource Info command from the Resource menu.
  3. Use Copy to put the ID number of the new 'PICT' in the scrap.

(Instead of steps 2, 3, and 7 here, you can always just read the ID number when you copy the 'PICT' and type it into the 'DITL' item by hand. ResEdit 2.0 displays the ID number of each 'PICT' resource.)

  4. Go to the 'DITL' that belongs to the 'DLOG' you are adding the picture to.
  5. Choose New Item.
  6. Click the PICT button.
  7. Paste the ID number from the scrap.
  8. Close the Dialog Item Editor.
  9. Choose Use RSRC Rect from the menu.
  10. Position the picture.

- If you are using the 'ICN#' editor or the 'ICON' editor, and you make a selection with the marquee and then cut or copy it, you can paste it as a 'PICT' resource. First make the type picker of your target (this can, of course, be the 'PICT' picker) be the active window. If you then paste, ResEdit makes the contents of its scrap into a new 'PICT'. The 'PICT' resource picker does *not* have to be open when you attempt to perform the paste operation.
- There are keyboard equivalents for many operations you would ordinarily perform with the mouse. Try selecting a file in the file open dialog by typing the first letter or two, then opening it with the Return key; you can do the same with resource types, and then with individual resources. (With individual resources, you can type the ID number or the name.) The arrow keys also work—for example, in a file list, you can go down the list with the down-arrow key.
- In general, it is a good idea to use the same ID for an 'ALRT' or 'DLOG' and its associated 'DITL', though this practice is not required.
- Other shortcuts and handy items:
  - In the resource picker: Option–double-click for Open Using Hex.
  - In the resource picker: Option–Command–double-click for Open Using Template.
  - In the resource picker: Option–Command–Shift–double-click (or Shift–Open Using Template) displays the template-type dialog box without the list of templates. (You can enter the template type you want.) If you are operating from a floppy disk, this can be a fast method.
  - Option-Cut and Option-Copy append the cut or copied item to the scrap. At the individual item editor level, holding down the Option key does not change the action of Cut or Copy.
  - In the 'DITL' editor: Option–Command–double-click on a 'CNTL', 'ICON', or 'PICT' to open it as a dialog item.
  - Command-click in a picker for disjoint selection.
  - Shift-click in a picker to extend a selection. (In a pictorial display such as the one for 'ICON' resources, the selection will extend as a rectangle.)
  - Using Shift-Create New Resource to create a new resource type gives you the "new type" dialog box without the list of resources. You must, of course, enter the resource type you want rather than selecting it from the list. If you are operating from a floppy disk, this can be a fast method.
  - In the bit editors ('CURS' and 'ICN#', for example), Shift-drag creates a selection rectangle (marquee). Using Shift-drag inside the marquee moves it. Releasing the Shift key and clicking inside the editing area turns off the marquee, but also inverts a bit in the picture. The marquee is also available in the 'FONT' editor.

- If you hold down the Command, Option, and Shift keys while choosing About ResEdit from the Apple menu, you can toggle a special stress-testing mode ("Pig mode"). In this mode, ResEdit performs a compact-memory operation and a purge-memory operation each time it receives an event from the queue, excepting null events. This feature was designed as an aid to debugging ResEdit itself, and is, clearly, something most people will never have any use for. It is suggested that you avoid invoking this mode unless you are writing an editor and need to stress-test it.
- If the 'DITL' for a 'DLOG' that is being displayed contains a reference to a 'CNTL' that doesn't exist, the editor will hang (in `NewDialog`) when it tries to draw the dialog box. Please be careful!
- Because 'DITL' and 'ALRT' resources are ordinarily displayed where you put them in the window, there is some chance that they may be mispositioned. That is, if you don't have your code display these resources exactly where you want them, they could show up where you *don't* want them. To be sure that a dialog box shows up where you want it, mark it as invisible and reposition it exactly in your code. Have your code mark it visible right after displaying it. (This avoids various embarrassments.)
- If you hold down the Option and Command keys and choose About ResEdit from the Apple menu, you get a list of credits that tells you who has worked on the program. Under MultiFinder, hold down the Option and Command keys, pull down any menu other than the Apple menu, and then move over to the Apple menu. Choose About ResEdit.
- Although under ordinary conditions the menu ID number and the 'MENU' resource ID' are kept identical to one another, there is one situation in which you may want to make them different. If you are using an ordinary debugger to disassemble and walk through the main event loop of your program, it is convenient to have the menu manager return numbers like 1, 2, 3, 4, and 5 for the menus in your program. You would therefore set the menu ID fields of your menus to consecutive integers. Then you might create a 'MBAR' resource with ID 128 and list the 'MENU' resource IDs of your menus in it. You need only call `GetNewMBar (128)` in your program to install all of the menus. When you are debugging, a call to `MenuSelect` (for example) returns a value of $00030004 if the 4th item in the 3rd menu has been chosen. This is rather more convenient than seeing $00820004 and having to translate $82 to 130 decimal, and then remembering that 130 was your third menu. If you use a high-level debugger this approach is unnecessary.

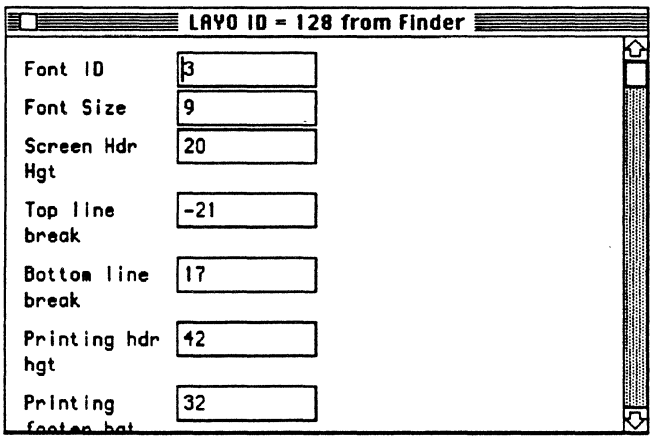
The 'LAYO' resource

One of the resources inside the Finder is of particular interest, because it controls a number of defaults, most of which are part of the layout of your desktop. It is the 'LAYO' resource. To open the Finder with ResEdit, you must be running under the Finder itself (rather than under MultiFinder), or you must edit a copy of the Finder. It is, of course, suggested that you edit a copy. If MultiFinder is running and you try to open the currently active Finder, you get an error message telling you that the Finder is already open from another application.

If you are in a risk-taking mood (or if you have done this a few hundred times already and have become inured to it), boot without MultiFinder, open the Finder, and choose the 'LAYO' resource type. There is only one 'LAYO' resource, ID number 128. Open it.

The first part of the template is shown in Figure 6-1.

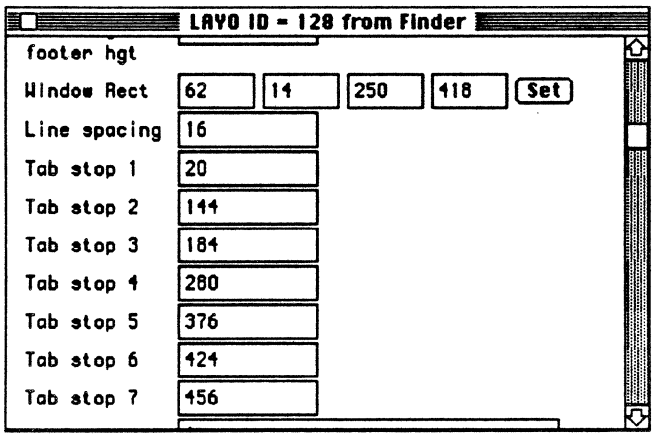
■ **Figure 6-1** 'LAYO' template, view 1



The first two items control the display font—that is, the font that prints out under the icons on your desktop. The default is 9-point Geneva, as shown. If you dislike sans-serif fonts, you can easily change the first two items to 2 and 9, for New York at 9 points, or to 20 and 10 (or even 12), for Times at 10 or 12 points; the 9-point version of Times is very small.

The line of numbers labeled Window Rect in Figure 6-2 allows you to specify the default folder (and disk) window size and location.

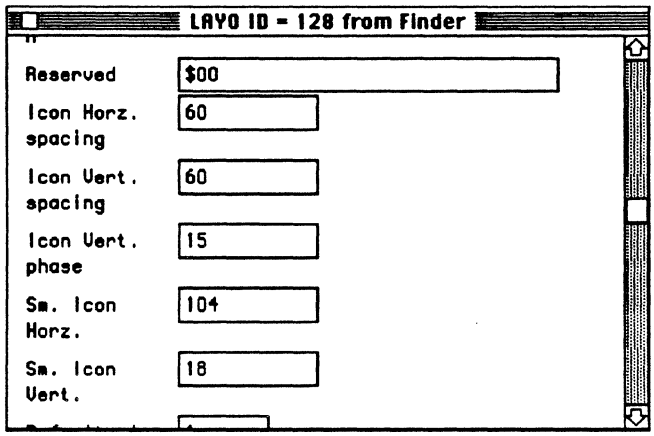
■ **Figure 6-2** 'LAYO' template, view 2



If you like, you can specify these defaults by clicking the Set button and then drawing a rectangle on the screen. Please note that if MultiFinder is running when you edit the 'LAYO' resource in a copy of the Finder, and you try to start your rectangle in an area of the screen that has something other than a ResEdit window in it, you will find yourself summarily ejected from ResEdit into whatever you have clicked. The cure is straightforward: Move a ResEdit window to the area where you want to start drawing your rectangle before you click the Set button, or use the number fields instead of the Set button. You can also explicitly set the locations of the seven tab stops the Finder uses for displaying information about files when you choose to view by Name, Date, Size, or Kind.

A bit further down the template are the numbers that control the placement of the icons themselves, as shown in Figure 6-3.

■ **Figure 6-3** 'LAYO' template, view 3

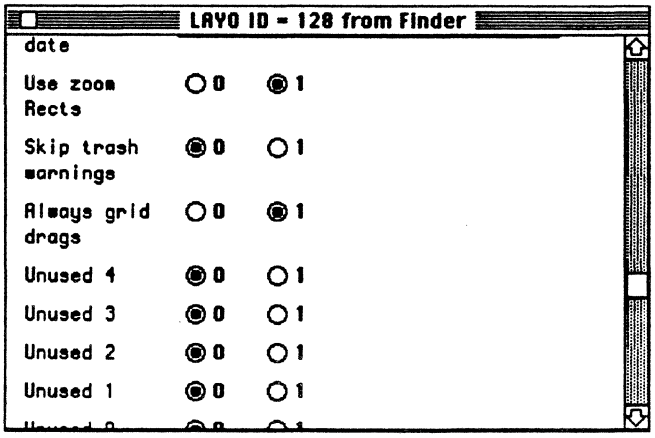


Some people dislike having icons with long names overlapping and obscuring the names of other icons. One solution to this problem is to reset the Icon Vertical phase. Figure 6-3 shows some modified numbers, rather than the defaults supplied with the system release.

▲ **Warning** Do not set the Icon Vertical phase to exactly half the Icon Vertical spacing unless you like system crashes. ▲

Figure 6-4 shows some unused bits and three commands, the first of which ("Use zoom Rects") is on by default. If you set it to False, the Finder will open and close windows slightly faster, because it won't use its 'zoom' visual effect.

■ **Figure 6-4** 'LAYO' template, view 4



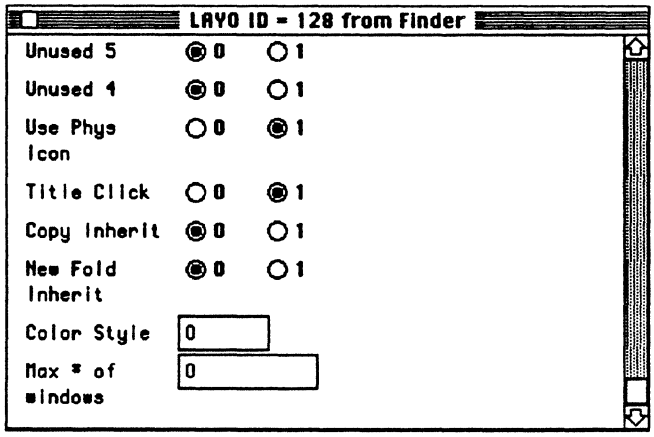
Skip trash warnings prevents the system from asking whether you really want to throw away that application or System file. Since you can avoid the warnings by simply holding down the Option key when you throw things into the trashcan, this seems a bit extreme. Moreover, it can be quite dangerous, depending on what you tend to throw out and how attentive you are about it.

If you don't like having to clean up your windows, try turning on Always grid drags. This option makes the icons stick in place at the grid spacing specified in the part of the template shown in Figure 6-3. Some people prefer to be able to put them anywhere and therefore eschew this option.

The Watch Thresh setting (not visible in any of the figures) allows you to adjust how long the Finder will wait during lengthy operations such as file copying before it displays a wristwatch cursor with animated hands. The time is expressed in 60ths of a second. If you make it too short, the cursor will jitter and change shape too often. Some older Finders do not make use of this option.

Figure 6-5 shows a few more unused bits and the end of the template.

■ **Figure 6-5** 'LAYO' template, view 5



Use Phys Icon is handy if you have a Macintosh II or Macintosh SE with two floppy disk drives. If this option is on, the icon you get when you insert a floppy disk into your machine indicates which drive the floppy disk is in. The disk location is certainly easy enough to recall just after you put the disk in, but you may forget it later. Knowing which drive a floppy disk is in may not be a major issue, but is certainly a pleasant convenience. This option also includes distinctive icons for an external hard disk and a CD-ROM drive.

Title Click lets you double-click the title bar of a folder's window to bring the parent folder's window to the front (or to open it if it is not already open). This feature can be quite handy.

When you create folders on an AppleShare® server, New Folder Inherit causes them to get their privileges from the parent folder, and when you duplicate existing folders on an AppleTalk server, Copy Inherit causes the copies to inherit their privileges from the originals.

The Max # of windows field allows you to set the maximum number of windows the Finder can have open at any one time. Increasing this number causes the Finder to need more memory. Under MultiFinder, you may have to increase the memory allocation for the Finder if you make this number much larger than the default.

Some of the items in the 'LAYO' template have not been discussed here. Of these, some are not yet in use. Others are either arcane or self-evident.

'KCHR' questions and answers

- How do I change the character generated by Shift-e?

Shift-e normally generates a capital E character. To make this key combination generate a different character, simply hold down the Shift key and use the mouse to drag a character from the character chart to the e key on the keyboard.

You will notice that when you press the Shift key, the table that is highlighted in the table list changes. (For most key layouts, the highlight switches from Table 0 to Table 1.) This change shows you that any character changes you make will be made in the highlighted table. When you make Shift-e generate a different character, you are changing every modifier key combination that uses the highlighted table. For example, if Option-Shift used the same table as Shift, you would also have changed the character generated by Option-Shift-e.

- How do I change the behavior of a modifier key combination?

For example, suppose you wanted Option-Shift-a to generate a different character from that generated by Option-Command-Shift-a. If you hold down the Option and Shift keys and then press and release the Command key, you will notice that (for most key layouts) the highlighted table does not change. If you want these two modifier key combinations to be different, you need to create a new table for one of them. To do this, you can use either the New Table command or the Duplicate Table command from the KCHR menu. If you want to create only a few differences, you should use the Duplicate Table command. In our example, we only want Option-Command-Shift-a to be different, so we would do the following:

1. Press and hold down the Option, Command, and Shift keys.
2. Choose Duplicate Table from the KCHR menu.
3. Select the new table that was added to the end of the list (while still holding down the modifier keys).
4. Choose OK in the alert box that appears.
5. Drag the character from the character chart to the key that you want to change (while still holding down all of the modifier keys).

- How do I remove a table that is no longer being used?

If you have reassigned a modifier key combination so that a table is no longer used, you can remove the table by choosing "Remove unused tables" from the KCHR menu. If there are unused or duplicate tables present when you close the editor, you will be asked whether they should be removed.

- How do I create a dead key?

You can create a dead key (such as Option-e in most key layouts) by choosing "Convert to dead key" from the KCHR menu while holding down the key. For example, follow these steps to make Option-k into a dead key:

1. Press and hold down the Option and k keys.
2. Choose "Convert to dead key" from the KCHR menu.
3. Release the keys.
4. Once again, press Option and k to activate the dead-key editor.

- How do I remove a dead key?

Follow these steps:

1. Select the dead key to display the dead-key editor.
2. Choose "Remove dead key" from the KCHR menu.

- How do I create a new completion/substitution pair in the dead-key editor?

When the dead-key editor is active, you can drag characters from the character chart to the completion/substitution pair list. The character on the left in the list is the completion character, and the character on the right is the substitution character. For example, Option-E produces the É character.

n How do I delete a completion/substitution pair in the dead-key editor?

To delete a completion/substitution pair, drag either character from that pair in the completion/substitution pair list to the trashcan in the lower-right corner of the window.

Chapter 7 **The Programmatic Interface**

You may want to create and edit your own types of resources. You can write pickers and editors as extensions to ResEdit in Pascal or C, and put them in the ResEdit Preferences file in your System Folder. This chapter describes this process and discusses necessary and optional functions and procedures.

Pickers and editors

Pickers and editors are separate from ResEdit's main code and hence may be supplied by user-written software.

The *picker* is the part that displays all the resources of your type in the resource type window. It is given the resource type and should display all resources of that type in the current resource file, using a suitable display format. If the picker is given an open call and there is a suitable editor, it should launch that editor. You need not supply your own picker; if a custom picker is not available, the standard picker is used to show a list of your resources with their names and IDs.

The *editor* is the code that displays and lets you edit a particular resource. The editor is given a handle to the resource object and should open an edit window for you.

Note that pickers and editors can be opened from anywhere in ResEdit or in your code. For instance, a dialog editor may open an icon picker so that you can choose an appropriate icon.

Code-containing resources in the ResEdit release

ResEdit includes three different types of resources that contain code. Much of the code is in the normal 'CODE' resources. The editors and pickers are found in the 'RSSC' resources, and the LDEF (or list definition) procedures are found in the 'LDEF' resources. The resource names of the pickers and editors are very important. The resource name of the 'RSSC' resource for a picker should be the resource type that the picker will pick. The resource name for an editor should be the resource type that the editor will edit, with a commercial "at" sign (@) in front of it. Subeditors (described in the section "Launching routines" later in this chapter) should have a dollar sign ($) in front of the resource type name. For example, the 'DITL' picker can be found in an 'RSSC' resource with the name DITL. The 'DITL' editor can be found in an 'RSSC' resource with the name @DITL, and the 'DITL' subeditor in an 'RSSC' resource with the name $DITL.

Samples

A sample resource editor, picker, and LDEF are included with ResEdit. The samples are provided in both C and Pascal and use the MPW 3.1 environment, the MPW C or Pascal Compiler, and the MPW Assembler. The appropriate build files and makefiles are also provided.

Sample editor

A sample ResEdit editor is provided in the file XXXXEdit. In this sample, *XXXX* represents your resource type. The sample editor will simply display a window and invert its contents. Since the details of editing your resource are known only to you, it is up to you to fill in the code necessary to make this sample into a real editor.

The sample editor is initialized by means of the `EditBirth` procedure when a resource of type XXXX must be edited. `EditBirth` is passed two handles: a handle to the resource to be edited (the same handle that would be received by using a `GetResource` call) and a handle back to the picker that launched the editor.

The editor then creates a window and sets up any data structures needed to operate. Because it may be loaded in and out of memory during any given session and because it doesn't have access to global variables, it creates a handle to a data structure to hold all data that needs to be preserved between calls. It stores the handle in the edit data structure `rXXXXRec`. Note that the handle to the edit data structure is stored in the window's `refCon` parameter. ResEdit uses this data structure to identify which editor or picker is to receive a given event.

ResEdit determines which editor should receive which events, so you need to worry only about events that affect your editor. During an update event, the `BeginUpdate` and `EndUpdate` calls are done by ResEdit, not by the extension program.

Sample picker

A sample ResEdit picker is provided in the file ICON.Pick. The sample picker is the actual 'ICON' picker from ResEdit. The 'ICON' LDEF (in the file ICON.LDEF) is included with this example so that you can see the interaction between a picker and its LDEF.

Sample LDEF

A sample ResEdit LDEF is provided in the file ICON.LDEF. An LDEF is a list definition procedure used to customize the way the List Manager draws and highlights cells. For more information, see *Inside Macintosh*, Volume IV, Chapter 30, and *Technical Introduction to the Macintosh Family*, Chapter 3. In ResEdit, LDEFs are used to customize the look of the picker windows. LDEFs are generally very simple procedures that draw or highlight a single cell of a list. The sample LDEF is the 'ICON' LDEF from ResEdit. This LDEF is used to display a file's Icons.

Building the examples

You can build the examples by using the build scripts provided in the folder appropriate to the language that you are using. The build scripts assume that ResEdit and the Examples folder will be found in the directory {boot}ResEdit:. If these files are located elsewhere, the build script files should be modified accordingly.

If ResEdit is successfully located, the MakeFile instructions will install the editor, picker, and LDEFs directly into ResEdit. When you experiment with changing any of these files, you may want to build into a copy of ResEdit. If anything goes wrong, you can then get a fresh copy of ResEdit to continue your experiments.

Using ResEd

The program you write must be a Pascal unit or C header file and library. Its interface with ResEdit is established by the MPW unit ResEd, contained in the file ResEd.p or ResEd.h. If your unit is written in PASCAL, it must begin with a `USES` declaration for this unit.

The assembly-language code that "opens up" ResEdit and activates your program is contained in the file RSSC.a. It must be linked with your Pascal or C module. When you open a resource of your type, ResEdit will call this code.

If your build script does not automatically install your editor or picker, place it in ResEdit's file by using ResEdit itself, with the type 'RSSC' and a unique ID number. Please use an ID number greater than 10,000 to avoid future conflicts. Your editor's name in the ResEdit file must be of the form @ABCD, where *ABCD* is the name you have assigned to the new type it edits. Install your picker (also of type 'RSSC') with the name ABCD (without the commercial "at" sign).

Writing a ResEdit extension

Here are two things to remember when writing a ResEdit extension:

- Always know which resource you are requesting and where it will come from. Many resource files may be open at any given time. Whenever a resource is needed, make sure which resource file you are accessing by using `UseResFile` or similar operations.
- Your editor may be called with an empty handle in order to create an entirely new instance of the type you edit.

In all of these procedures, remember to lock any handle that is going to be dereferenced (for example, in a Pascal `with` statement). For example, in Pascal, the first instructions in the `DoEvent` procedure should be

```
BubbleUp(Handle(object));
HLock(Handle(object));
```

It is important to call the `BubbleUp` procedure to avoid heap fragmentation. Remember to unlock the object at the end of the procedure!

If any of these procedures will need access to the current port, especially `EditBirth`, `DoEvent`, and `DoMenu`, call

```
SetPort (object^^.wind)
```

if you are writing in Pascal, or

```
SetPort ((*object)->wind)
```

if you are writing in C.

ResEdit 2.0 changes

Here's what you have to do to upgrade an editor to ResEdit 2.0:

- Change the name field of your parent record from `STR64` to `STR255`.
- Add `AbleMenu` for the Resource menu on activate:

```
AbleMenu (rsrcMenu, rsrcEditor);
```

- Change `AbleMenu` for the File menu to

```
AbleMenu (fileMenu, fileAll);
```

- Add `PrintItem` to the `DoMenu` procedure:

```
printItem:
        PrintWindow (NIL);
```

- In `DoMenu`, change `RevertItem` to `rsrcRevertItem` and `GetInfoItem` to `rsrcGetInfoItem`. Move them from the File menu to the Resource menu.
- Add the `IsThisYours` function and be sure to make it public. See the example code for details.
- `EditorWindSetup` now requires a `windowKind` parameter and a `dlogID` parameter; `windowKind` should be the resource ID of the editor or picker (returned by ResEdID), and `dlogID` should be NoDialog or the resource ID of a dialog to be used for the window.
- `WindOrigin` now takes a `ParentHandle` parameter and requires that the `windowKind` field of the argument window be set to the resource ID of the editor.

ResEd changes for the 2.0 release

- `PickRec` was changed to remove some unused fields and add other fields for the View menu.
- `ParentRec` was changed to include an STR255 instead of STR64.
- Menu and string constants were changed.

Several procedures have Interface changes; these are the new interfaces:

```
        FUNCTION  EditorWindSetup (dlogID: INTEGER; color: BOOLEAN; width,
height: INTEGER; VAR windowTitle, windowName: STR255; addFrom: BOOLEAN;
windowKind: INTEGER; father: ParentHandle): WindowPtr;
        PROCEDURE WindOrigin (w: WindowPtr; dad: ParentHandle);
```

```
        PROCEDURE PickMenu (tossOnClose: BOOLEAN; menu, item: INTEGER;
pick: PickHandle);
```

The following routines are no longer available:

```
CWindSetup
WindSetup
RevertResource
PickStdRows
CallEBirth
CallEvent
CallMenu
CallPBirth
CopyRes
DoKeyScan
DoListEvt
```

Required routines

Each picker and editor must contain a set of required procedures. Some of these procedures are appropriate only for editors, and others are appropriate only for pickers, but all of them must appear in all editors and pickers.

EditBirth

```
PROCEDURE EditBirth (theResource: Handle; dad: ParentHandle);
```

This procedure should initialize the editor data structure and create an editor window for the given resource type. In a picker, this procedure will do nothing and should be defined as

```
PROCEDURE EditBirth (theResource: Handle; dad: ParentHandle);
BEGIN
END;
```

PickBirth

```
PROCEDURE PickBirth (theType: ResType; dad: ParentHandle);
```

This procedure should initialize the picker data structure and create a picker window for the given type. `PickBirth` is very similar to `EditBirth` except that it takes a resource type as a parameter instead of a resource handle. The `DoPickBirth` procedure can usually be used to take care of most initialization for a picker. In an editor, this procedure will do nothing and should be defined as

```
PROCEDURE PickBirth (theType: ResType; dad: ParentHandle);
BEGIN
END;
```

DoEvent

```
PROCEDURE DoEvent(VAR evt: EventRecord; object: ParentHandle);
```

`DoEvent` handles all events for the picker or editor. The object parameter can be locally defined as whatever type is appropriate (such as a `PickHandle`) instead of the generic `ParentHandle`.

Editors will normally handle all of the events (except those described in the next paragraph) themselves, whereas pickers should simply call `PickEvent`.

Many events are handled by the main part of the ResEdit code before the `DoEvent` procedure is called. For mouse-down events, ResEdit handles the following events: pulling down menus, dragging windows, switching between windows, and converting doubleclicks to open commands. Update events call `BeginUpdate` and `EndUpdate` around the call to `DoEvent`. For key-down events, the `DoMenu` procedure is called if the Command key was down (unless the key was Return, Enter, or an arrow); `DoEvent` is called otherwise. MultiFinder suspend and resume events are converted into the appropriate activate or deactivate events.

DoInfoUpdate

```
PROCEDURE DoInfoUpdate(oldID, newID: INTEGER; object: ParentHandle);
```

This procedure is called when information about a resource—for example, its ID number—is changed in a Get Info window. (See the `ShowInfo` procedure, discussed later in this chapter in the section "Miscellaneous utilities.") For editors, the `DoInfoUpdate` procedure should recalculate the window title and the name stored in the `ParentHandle` and pass the update on to its father by using the `CallInfoUpdate` procedure as follows:

```
      CallInfoUpdate(oldID, newID, object^^.father^^.wind^.refCon,
object^^.father^^.wind^.windowKind);
```

Pickers should simply call

```
      PickInfoUp (oldID, newID, object);
```

DoMenu

```
PROCEDURE DoMenu(menu, item: INTEGER; object: ParentHandle);
```

`DoMenu` handles all menu events for the picker or editor. The object parameter can be locally defined as whatever type is appropriate (such as a `PickHandle`) instead of the generic `ParentHandle`.

The main part of the ResEdit code takes care of several of the menu-handling details. All selections from the Apple menu are handled so that the editors and pickers do not need to know anything about desk accessories. All commands in the File menu are also handled for you. The Quit command displays the Save Changes dialog box and may pass a Close command to all editors and pickers. If your editor needs to do some cleaning up before the Quit command completes, it should do so when it receives a Close or deactivate command. If "no" is chosen in the Save File dialog box, the frontmost window receives a deactivate event. No events are passed to any other window. When your editor receives a Close command, it can call `CloseNoSave` to see whether edit checking should be performed. If the current file is being closed but the changes are not being saved, `CloseNoSave` will return TRUE, and edit checking should not be performed.

Pickers can simply call

```
PickMenu (tossOnClose, menu, item, object);
```

If your picker has loaded all of the resources, it should call `PickMenu` with `tossOnClose` set to TRUE so the resources are released when a Close command is received.

Using custom LDEFs

You will typically want to write your own picker simply to display the resource list in a more meaningful way (such as drawing the icons themselves in the 'ICON' picker, instead of listing their names). You can easily accomplish this task by providing a simple picker and a custom LDEF that is used for drawing the picker list. When you call `DoPickBirth` in your `PickBirth` procedure, pass the resource ID of your picker as the `PickerResID` parameter. You can get the resource ID by calling `ResEdID`. The resource ID is passed on to the `LNew` procedure. You should then provide a custom LDEF with the same resource ID. The LDEF will be called whenever the list needs to be updated. Please refer to the chapter on the List Manager in *Inside Macintosh,* Volume IV, for details of the workings of the `drawProc` mechanism.

In most cases, the DrawLDEF procedure takes care of most of the tasks required of an LDEF. All you have to do is provide a procedure to draw your resource type.

The ResEd interface

The ResEd unit contains data structures, procedures, and functions that you can access from your extension program. They are described in the remainder of this chapter.

Data structures

The ResEd unit declares the data structures described in this section, which provide communication between extension programs and ResEdit. Each editor or picker has its own object handle. The data structure has to start with a handle to its parent's object, followed by the name distinguishing the father. This name will be part of the son's window title. The next field should be the window of the object that may be used by the son to get back to the father through the `refCon` in the `windowRec` record. The next field is the rebuild flag, which is used to indicate that a window's data (for example, a picker's list) must be recalculated at the next opportunity. For editors, the rest of the handle can have any format; pickers have additional data, as described in this chapter. Editors and pickers typically declare additional fields following the rebuild field, and can store in these additional fields global data that they need to access from the `DoEvent`, `DoInfoUpdate`, and `DoMenu` procedures.

The name (in the `ParentRecord`) for a picker should be the name of the file, folder, or disk. For editors, the name should be the complete name (not the window's title), preceded by an `editorNameChr` character. An example of a complete name would be "ALRT ID = -1234 from AFile". This name is used to uniquely identify a window. The window's title is created by `GetWindowTitle` or `EditorWindSetup`, described later in this chapter.

- *Note:* It is important for editors and pickers to follow these conventions for name and window title. For pickers, it is more important that the window's title be unique, and for editors, that the name be unique. The `AlreadyOpen` procedure uses the window's name and title to determine whether the window is open. Please refer to the description of `AlreadyOpen` later in this chapter in the section "Window management routines" for complete information about how the name and title are used.

The parent record

```
ParentPtr = ^ParentRec;
ParentHandle = ^ParentPtr;
ParentRec = RECORD
    father: ParentHandle;     { Back ptr to dad }
    name: Str255;
    wind: WindowPeek;
    rebuild: BOOLEAN;         { Flag set by son to indicate that }
                              { world has changed so father should }
                              { rebuild list }
    END;
```

The picker record

The record for pickers is slightly different from the standard parent record. The first four fields are the same as those in the parent record. The rest of the fields are specific to pickers.

```
PickPtr = ^PickRec;
PickHandle = ^PickPtr;
ViewTypes = (viewById, viewByName, viewBySize,
             viewByOrder, viewBySpecial);
PickRec = RECORD
   father:          ParentHandle;     { Back ptr to dad }
   fName:           STR255;
   wind:            WindowPtr;        { Directory window }
   rebuild:         BOOLEAN;          { Flag set by son to
                                        indicate that father
                                        should rebuild list }
   pickID:          INTEGER;          { ID of this picker }
   rType:           ResType;          { Type for picker }
   rNum:            INTEGER;          { Resfile number }
   rSize:           LONGINT;          { Size of a null resource }
   nInsts:          INTEGER;          { Number of instances }
   instances:       ListHandle;       { List of instances }
   drawProc:        Ptr;              { List draw proc }
   scroll:          ControlHandle;    { Scroll bar }
   viewBy:          ViewTypes;        { Current view type }
   ldefType:        ResType;          { Which LDEF to use }
   theViewMenu:     MenuHandle;       { The picker view menu }
   showAttributes:  BOOLEAN;          { Show attrs in window? }
   viewMenuMask:    LONGINT;          { Which items are enabled? }
   cellSize:        Cell;             { Cell size for special view. }
END;
```

Other routines

The required routines are called by ResEdit itself. Here are others you can use. These are called by the editor or picker.

Launching routines

```
PROCEDURE GiveEBirth (resHandle: Handle; pick: PickHandle);
```

`GiveEBirth` starts an editor. This routine is used when a picker wants to start an editor or when an editor wants to start another editor (as when the 'DLOG' editor starts the 'DITL' editor). If Open Using Template was chosen or an editor is not found, the 'GNRL' (template) editor is started. If Open Using Hex Editor is chosen or neither an editor nor a template is found, the hexadecimal editor is started. A call to the appropriate editor's `EditBirth` procedure is then generated, as follows:

```
EditBirth (resHandle, pick)
```

In this call, `ResHandle` is the handle of the resource that is to be edited, and `Pick` is the caller's `ParentHandle`.

- *Note:* When an editor is starting another editor, it is important to remember that `pick^^.rType` and `pick^^.rNum` must be set before this routine is called. The editor's `ParentRecord` will need to be equivalent to a `PickRec`, at least down to the `rNum` field. The `GiveEBirth` procedure looks into the `PickHandle` parameter for information (for example, the resource type) that it needs to start up an editor.

```
PROCEDURE GiveThisEBirth (resHandle: Handle; pick: PickHandle;
                          openThisType:ResType);
```

`GiveThisEBirth` is similar to `GiveEBirth`, except that it lets the caller specify the type of editor to open. The specified editor is opened even if Open Using Template or Open Using Hex Editor is chosen. If an editor of the specified type is not found, a template of the specified type is opened. If a template is not found, the hexadecimal editor is opened.

```
PROCEDURE GiveSubEBirth (resHandle: Handle; pick: PickHandle);
```

`GiveSubEBirth` starts an editor that edits a part of another type of resource. For example, the 'DITL' editor uses `GiveSubEBirth` to start the Dialog Item Editor. `GiveSubEBirth` behaves exactly like `GiveEBirth` except that the name of the resource that it looks for begins with a dollar sign ($) instead of a commercial "at" sign (@). For example, the name of the 'DITL' editor resource is @DITL and the name of the 'DITL' subeditor resource is $DITL. This distinction allows an editor to use the standard method for editing multiple occurrences of a subtype within the resource. For example, a dialog item list ('DITL') typically contains several dialog items. Calling `GiveSubEBirth` lets the user open multiple dialog items and treat them in the same way as any other windows.

Information-passing routines

```
PROCEDURE CallInfoUpdate (oldID, newID: INTEGER;
  refcon: LONGINT; id: INTEGER );
```

`CallInfoUpdate` passes an information update command to the specified window. After updating its own window and data structures, each editor's `DoInfoUpdate` procedure should call this routine to pass the information update along to its parent window. This call is necessary since the parent may be displaying data (such as the ID or name in a picker window) that has been changed. An editor could pass this information along by making the following call:

```
CallInfoUpdate (oldid, newid, father^^.wind^.refcon,
father^^.wind^.windowkind);
```

```
PROCEDURE PassMenu (menu, item: INTEGER; father:
  ParentHandle);
```

`PassMenu` passes menu commands on to any son pickers or editors that you have started. For example, when your editor receives a Close command, it should pass that command along to any subeditors or information windows that it has opened by making the following call:

```
PassMenu (fileMenu, closeItem, myObj)
```

Window management routines

```
FUNCTION AlreadyOpen (VAR windowTitle, windowName: STR255;
  dad: ParentHandle): BOOLEAN;
```

`AlreadyOpen` looks to see if the window is already open. If the window is open, `AlreadyOpen` activates it and returns TRUE. `WindowTitle` and `windowName` are as defined in the note immediately below. You don't need to call this function if you are using the `PickerWindSetup` or `EditorWindSetup` procedure.

- *Note:* You should call `AlreadyOpen`, to avoid opening the same resource twice. `AlreadyOpen` depends on your setting `windowTitle` and `windowName` correctly. For pickers, the window's title must uniquely identify the window. For editors, the name stored in the parentRec data structure must uniquely identify the window. The name is used for editors so that the window title can be simple and short. For example, the window title for a dialog item might be `Edit DITL item #3`, whereas its name would be `Edit DITL item #3 • DITL "<resource name>" id = <num> from <file name>`.

```
PROCEDURE GrowMyWindow (minWidth, minHeight: INTEGER;
  windPtr: WindowPtr; lh: ListHandle);
```

This procedure is used by pickers to grow their windows. The `minWidth` and `minHeight` parameters determine the minimum size of the window; `windPtr` is the window to be grown; `lh` is the list that is in the window.

The `GrowMyWindow` procedure takes care of everything that is necessary to grow a picker's window. If necessary, the list is resized and redrawn. Two-dimensional lists (such as those used by the icon picker) are updated to fit as many cells as possible in the window without requiring horizontal scrolling.

```
PROCEDURE GetWindowTitle (VAR windowTitle, windowName: STR255;
  addFrom: BOOLEAN; dad: ParentHandle);
```

`GetWindowTitle` constructs the window title and name for an editor. This routine should always be called in the `DoInfoUpdate` procedure, and should be called in the `EditBirth` procedure if `EditorWindSetup` is not called. `WindowTitle` should be used for the window's title. `AddFrom` determines whether or not the name of the file is added to the title. `WindowName` should be saved in the name field of the editor's data structure. This name is used later to identify the window uniquely. On input, `windowTitle` should contain only the title or the resource (for example, 'ALRT'), and `windowName` should contain the resource type (for example, 'ALRT'). If `EditorWindSetup` is not used, the following code fragment can be used to assure that the name and title are correct:

```
GetResInfo(myResource, theID, theType, windowTitle);
TypeToString (theType, windowTitle);
SetETitle(myResource, windowTitle);
windowName := windowTitle;
GetWindowtitle (windowTitle, windowName, TRUE, parent);
```

```
PROCEDURE SetETitle (resHandle: Handle; VAR title: STR255);
  Extended Resource Manager
```

`SetETitle` concatenates the resource's ID with its name and places the result into `title`. The `resHandle` parameter is the handle to the resource. You can use this routine when you are constructing a window's name or title.

```
FUNCTION WindAlloc: WindowPtr;
```

`WindAlloc` returns a pointer to a window record to be used by your editor or picker. Using this routine instead of allocating your own window pointer can help reduce heap fragmentation. Because windows are pointers and must be nonrelocatable objects in the heap, ResEdit uses this procedure to try to allocate `WindowPtr` pointers as low in the heap as possible. When this procedure is called, it usually returns a `WindowPtr` that it has previously allocated low in the heap.

```
PROCEDURE WindReturn (w: WindowPtr);
```

`WindReturn` returns a window pointer that was allocated by `WindAlloc`. Use this procedure when you terminate your editor or picker and you are finished with its window. `WindReturn` makes the memory used by the window available to another picker or editor for use as a new window. This helps keep the nonrelocatable window pointers as low in the heap as possible.

```
FUNCTION WindList (w: WindowPtr; nAcross: INTEGER;
  cSize: Point; drawProc:INTEGER): ListHandle;
```

`WindList` creates a new empty list and returns a handle to that list. This procedure should be used by pickers to allocate their lists. `WindList` calls the `LNew` procedure to allocate a list. `w` is the window in which the list will be created. `nAcross` specifies the number of cells across that the list should contain. The list is allocated with 0 rows. `cSize` is the `cSize` parameter to `LNew`. `drawProc` is the `Proc` parameter to `LNew`. For more information on lists and a description of the `LNew` parameters, see the chapter on the List Manager in *Inside Macintosh*, Volume IV. Please refer to the section "Using custom LDEFs," earlier in this chapter, for information on specifying custom draw procedures.

```
PROCEDURE WindOrigin (w: WindowPtr; dad:ParentHandle);
```

`WindOrigin` moves the window pointed to by `w` to the first available position in the set of offset positions; this is usually a position immediately below and to the right of the front window. If `w` is a color window, the window is positioned on the deepest available display device. This routine guarantees that, if possible, the entire window will be visible. `WindOrigin` requires the windowkind field of `w` be set to a ResEdit value (for example by a call to `ResEdId`), and that the window size be set. If you are using the `PickerWindSetup` or `EditorWindSetup` procedure, you don't need to call this procedure.

```
FUNCTION  PickerWindSetup(color: BOOLEAN; width,
  height: INTEGER; VAR windowTitle: STR255;
  windowKind: INTEGER; dad: ParentHandle): WindowPtr;
```

PickerWindSetup should be called by pickers from the PickBirth procedure. It is similar to the EditorWindSetup procedure.

```
FUNCTION EditorWindSetup (dlogID: INTEGER; color: Boolean;
  width, height: INTEGER; VAR windowTitle,
  windowName: STR255; addFrom: BOOLEAN;
  windowKind: INTEGER; dad: ParentHandle): WindowPtr;
```

`EditorWindSetup` should be called by editors from the `EditBirth` procedure to set up their windows. If the `color` parameter is `TRUE`, a color window is returned. Color windows are positioned on the deepest available display. `WindowTitle`, `windowName`, and `addFrom` are passed directly to `GetWindowTitle`. Refer to the description of `GetWindowTitle` for details about these parameters. `WindowName` is returned with the string that should be used for the name in the `ParentRecord`. This routine also takes care of constructing the `windowTitle` and `windowName` correctly so that the window can be uniquely identified. If dlogID is not set to noDialog, the width and height parameters should be set to 0 if you want to use the size stored in the DLOG resource. Use the `dlogID` parameter if you want your window to be a dialog; for normal windows, pass the constant `noDialog`. The `windowkind` parameter is used to initialize the window. Pass the result of a `ResEdId` call here.

- *Note:* NIL is returned if the window can't be allocated for some reason or the window is already allocated (that is, an editor is already open). If NIL is returned, the `EditBirth` routine should be aborted.

Resource utilities

```
FUNCTION AddNewRes (hNew: Handle; t: ResType; idNew: INTEGER;
  s: str255): BOOLEAN;
```

`AddNewRes` has the same parameters and performs the same actions as the Macintosh procedure `AddResource`. The only difference is that if an error is detected, an alert is displayed and FALSE is returned; TRUE is returned otherwise.

```
FUNCTION  BeautifulUnique1ID (t: ResType): INTEGER;
```

This routine should be used instead of the toolbox procedure Unique1ID. It will return the first unused resource ID starting with ID 128.

```
FUNCTION CurrentRes: INTEGER;
```

`CurrentRes` returns the ID number of the current resource file. This routine is the same as the `CurResFile` trap except that if `CurResFile` returns `SysMap`, this routine returns 0 (for the System file).

A typical use of this routine is to save the current resource file so that it can be restored later. For example:

```
savedResFile := CurrentRes;
UseResFile(someOtherRes);
.
.
.
UseResFile(savedResFile);
```

```
FUNCTION Get1Index (t: ResType; index: INTEGER): Handle;
```

`Get1Index` is similar to the `Get1IndResource` trap. The only difference is that if the resource is not found, this routine will set `ResError` to the `resourceNotFound` error and return NIL.

```
FUNCTION Get1Res (t: ResType; id: INTEGER): Handle;
```

`Get1Res` is similar to the `Get1Resource` trap. The only difference is that if the resource is not found, this routine will set `ResError` to the `resNotFound` error and return NIL.

```
PROCEDURE Get1MapEntry (VAR theEntry: ResMapEntry;
  t: ResType; id: INTEGER);
```

`Get1MapEntry` accesses the current resource map for a resource of type `t` and ID number `id`, placing the result in `theEntry`. For a description of resource maps, see "Format of a Resource File" in *Inside Macintosh*, Volume I, Chapter 5.

```
PROCEDURE Get1IMapEntry (VAR theEntry: ResMapEntry;
  t: ResType; index: INTEGER);
```

`Get1IMapEntry` is similar to `Get1MapEntry`, except that it refers to its resource by index instead of by ID number.

```
FUNCTION NeedToRevert (myWindow: WindowPtr; theRes: Handle):
  Boolean;
```

The `NeedToRevert` function should be called by all editors before they revert their resource. If the editor's window is the frontmost window and the resource has been changed, an alert is displayed asking the user to verify that he or she really wants to revert the resource. If the user does want to revert the resource, the function returns a value of TRUE. Otherwise it returns a value of FALSE. The `myWindow` parameter is a pointer to the editor's window. The `theRes` parameter is the handle of the resource that is to be reverted.

```
FUNCTION NewRes (s: LONGINT; t: ResType; l: ListHandle;
  VAR n: INTEGER): Handle;
```

Given a size, `s`, `NewRes` allocates a new handle, clears it, adds it to the current resource file as a resource of type `t` with a unique ID, adds it to the list `l` (unless `l` is NIL), and returns a handle to the new resource. The parameter `n` is the item number in the list `l`. If this function fails, it returns a NIL handle.

```
FUNCTION  ResEditGet1Resource (theType: ResType; ID: INTEGER;
  VAR wasLoaded: BOOLEAN; VAR error: INTEGER): Handle;
```

`ResEditGet1Resource` should be used in place of the toolbox routine `Get1Resource`. It's equivalent to `Get1Resource` except for the fact that it returns a `wasLoaded` variable to indicate whether the resource is already in use. If `wasLoaded` is returned TRUE, the caller should NEVER free the resource with the `ReleaseResource` procedure.

```
PROCEDURE ResourceIDHasChanged (theObj: ParentHandle;
  theType: ResType; theOldId, theNewId: INTEGER);
```

Call this procedure if you have changed the ID of a resource. If you change a resource ID and don't call this routine, revert won't work properly.

```
FUNCTION RevertThisResource (theObj: ParentHandle;
  res: Handle): BOOLEAN;
```

`RevertThisResource` restores a resource being edited to the state it was in before editing started. The parameter `res` is a handle to the resource. The parameter `theObj` is the `ParentHandle` from the current window. It is needed to determine whether the resource was newly added. The `RevertThisResource` function returns a value of FALSE if the resource was newly added by ResEdit (and, therefore, no longer exists after the reversion), and TRUE otherwise. If the resource has not been changed (its `resChanged` flag is not set), nothing is done.

```
PROCEDURE RemoveResource (theRes: Handle);
```

This procedure should always be used in place of the toolbox call `RmveResource`. It correctly handles resources that have the protected attribute set, by unprotecting them before removing them. The function of this routine is otherwise the same as that of the `RmveResource` toolbox procedure.

```
FUNCTION SysResFile: INTEGER;
```

This function returns the resource file ID of the System file. It is often necessary to take special precautions when accessing the System file. This function allows you to take these precautions without hard-coding a value for the system resource file ID, which may change in the future.

Miscellaneous utilities

```
PROCEDURE Abort;
```

`Abort` sets the abort flag, which will stop any command that is in progress. The most common use of this command is in stopping the Quit command. For example, if an error is detected in a template when its window is being closed, the template editor calls `Abort` so that processing of the Quit command will stop and the error can be corrected.

```
FUNCTION WasAborted: BOOLEAN;
```

`WasAborted` returns the state of the aborted flag (set by the `Abort` procedure just described). This function is useful, for example, if you have just called `PassMenu` with a Close command and you want to know if any of the windows that were closed encountered a problem.

```
PROCEDURE AbleMenu (menu: INTEGER; enable: LONGINT);
```

`AbleMenu` enables or disables menu items. `Ablemenu` differs from the Resource Manager routines `EnableItem` and `DisableItem` in that it acts on the entire menu. The parameter `menu` is a menu ID; `enable` is a mask. Values used for the mask can be found in the ResEd file.

```
PROCEDURE BubbleUp (h: Handle);
```

`BubbleUp` sets up the correct heap zone and then performs the Memory Manager routine `MoveHHI`. For information about `MoveHHi`, see *Inside Macintosh*, Volume II, Chapter 1. This routine should always be called, to avoid heap fragmentation, before the Macintosh procedure `HLock` is called for any handle. Remember to unlock any handle that you lock!

```
FUNCTION BuildType (t: ResType; l: ListHandle): INTEGER;
```

Given a list that has been initialized with no rows, `BuildType` builds a list of all resources of type `t` from the current resource file. (See the `WindList` routine described earlier in this chapter.) If `SetResLoad (FALSE)` has not been called, all of the resources will be loaded into memory. `BuildType` returns a count of the number of instances that it adds to the list.

A picker that doesn't use `PickerWindSetup` can set up its window with this sequence:

```
myList := WindList(myWindow, myListWidth, myCellSize, ResEdid);
LDoDraw(FALSE, myList);                {draw it later}
NInsts := BuildType(myType, myList);
LSetSelect(TRUE, Cell(0), myList);     {automatically select first cell}
LDoDraw(TRUE, myList);                 {ok to draw it next time}
```

```
PROCEDURE CenterDialog (theType: ResType; dialog: INTEGER);
```

This procedure centers dialogs or alerts on the same screen as the current port, which is assumed to be a window. If the dialog is in color, it is centered on the screen with the most colors on which any portion of the current port appears. `ResType` can be 'DLOG' or 'ALRT'; `dialog` is the resource ID of the dialog or alert. The 'DLOG' or 'ALRT' resource is loaded into memory and its boundsRect is centered. When you use the dialog or alert (for example, in `GetNewDialog`) the resource will be found in memory with the correct boundsRect.

```
FUNCTION CheckError (err, msgID: INTEGER): BOOLEAN;
```

`CheckError` displays an error alert if `err` is nonzero. This routine has built-in alert messages for several errors (such as disk write-protected, out of memory, and so on). If `msgId` is negative, a fatal error message is retrieved from the 'STR#' resource with ID of 128. This resource is preloaded into memory, and may be accessible even if a serious error has occurred. If `msgID` is nonnegative, an error message from the 'STR#' resource with ID of 129 is displayed. If the error is not one that is built in, the string with an ID of `msgID` is displayed in the alert. TRUE is returned if `err` was zero, FALSE otherwise. When adding a new string for use by `CheckError`, be sure to add it to the end of the existing list in the 'STR#' resource.

```
FUNCTION CloseNoSave: BOOLEAN;
```

`CloseNoSave` returns a Boolean value that indicates whether data checking should be performed before closing. A return value of TRUE indicates that checking should not be performed. For example, if the user is editing a template and there are errors in the template when the Quit command is chosen, the template editor should not perform edit checking if "no" was clicked in the Save Changes dialog box.

```
FUNCTION  ColorAvailable (needColorQD: BOOLEAN): BOOLEAN;
```

`ColorAvailable` returns TRUE if color QuickDraw is available. If the `needColorQD` parameter is TRUE, an alert is displayed if color QuickDraw is not available.

```
PROCEDURE ConcatStr (VAR str1: STR255; str2: STR255);
```

`ConcatStr` concatenates `str2` to `str1`, leaving the result in `str1`.

▲ **Warning** This routine does not check for aggregate string lengths in excess of 255 characters. Please be careful! ▲

```
FUNCTION DefaultListCellSize:INTEGER;
```

`DefaultListCellSize` returns the height of a list cell with the application font (ascent + descent + leading). This function should be used by pickers that display resources as text strings when setting up their window.

```
FUNCTION DisplayAlert (which: AlertType; id: INTEGER):
INTEGER;
```

`DisplayAlert` displays an alert with the given `id`. This routine assures that the alert resource is loaded from ResEdit and that the cursor is reset to an arrow. The `which` parameter determines the kind of alert that is displayed.

```
AlertType = (displayTheAlert, displayStopAlert, displayNoteAlert,
displayCautionAlert);
```

```
FUNCTION  DisplaySTRAlert(which: AlertType; STRName: STR255;
  STRIndex: INTEGER): BOOLEAN;
```

This procedure is similar to `DisplayAlert` except that a standard alert box is used and the text is retrieved from a 'STR#' resource. If you want to display an alert, just create a 'STR#' resource in ResEdit and call this routine with the 'STR#' resource name and the index in the string list of the string to be used. Whenever possible, this routine should be used instead of `DisplayAlert`.

```
FUNCTION  DoPickBirth(color: BOOLEAN; buildList: BOOLEAN;
  wWidth, wHeight, columns: INTEGER; pickerResId: INTEGER;
  pick: PickHandle): BOOLEAN;
```

`DoPickBirth` takes care of just about everything needed to initialize a picker. If `buildList` is TRUE, the list of all of the resources will be created. `Pick` is the handle to a partially initialized `PickHandle`. The fields that should be initialized before this procedure is called are: `father`, `rType`, `viewBy`, `cellSize`, and `ldefType`. The example picker shows how these fields should be initialized.

```
PROCEDURE DrawLDEF (message: INTEGER; lSelect: BOOLEAN;
  lRect: Rect; theRes: Handle; id: INTEGER;
  title: STR255; maxH, maxV: INTEGER;
  DrawResource: ProcPtr; lh: ListHandle);
```

`DrawLDEF` is a general purpose drawing routine for graphical LDEFs like 'ICON', 'cicn', and so on. It should be called from an LDEF that is used by a picker. If `title` is an empty string, `id` is converted to a string and used as the title. The `drawProc` is of the form: `PROCEDURE DrawResource (lRect: Rect; theRes: Handle)`.
Use of this procedure is shown in the example picker LDEF.

```
PROCEDURE DrawMBarLater (forceItNow: BOOLEAN);
```

`DrawMBarLater` should be used instead of the toolbox `DrawMenuBar` procedure. It will collect updates to the menubar but actually draw the menubar only when no other events are pending. Using this procedure avoids flashing the menubar as menus are added and removed. If `forceItNow` is TRUE, the menubar is drawn immediately and any pending updates are cleared.

```
FUNCTION  FindOwnerWindow (theRes: Handle): WindowPeek;
```

`FindOwnerWindow` checks all of ResEdit's windows to see if an editor is open for the specified resource. If you're writing an editor that uses a resource that may be in use by another editor (for example, two 'DLOG' resources may share the same 'DITL'), call `FindOwnerWindow` to determine whether the resource should be released.

```
PROCEDURE FixHand (s: LONGINT; h: Handle);
```

`FixHand` makes sure that the object to which `h` is a handle is `s` bytes long. If it is longer, `FixHand` shrinks it; if it's shorter, `FixHand` expands it and fills the extension with zeros.

```
PROCEDURE GetNamedStr(index: INTEGER; name: STR255;
  VAR str: STR255);
```

`GetNamedStr` returns in `str` the `index`th string in the 'STR#' resource named `name`. All strings should be stored in either 'STR#' or 'STR ' resources to maintain the international localizability of ResEdit.

```
PROCEDURE GetStr (num, list: INTEGER; VAR str: STR255);
```

`GetStr` returns, in `str`, string number `num` from ResEdit's 'STR#' resource with ID of `list`. All strings should be stored in either 'STR#' or 'STR ' resources to maintain the international localizability of ResEdit.

```
PROCEDURE FlashDialogItem (dp: DialogPtr; item: integer);
```

`FlashDialogItem` flashes (inverts) a dialog button for 8 ticks to indicate that the button was selected. This procedure should be called from a dialog's filter procedure.

```
PROCEDURE FrameDialogItem (dp: DialogPtr; item: integer);
```

`FrameDialogItem` draws a frame around a dialog button to indicate that it is the default button (the button that will be selected when either the Return or the Enter key is pressed). This procedure should be called when an update event is received by a dialog's filter procedure.

```
FUNCTION GetQuickDrawVars: pQuickDrawVars;
```

This function returns a pointer to the QuickDraw variables that are normally available to Macintosh programmers. Because of the way that pickers and editors are implemented, they do not normally have access to these variables. The following types are used with this function:

```
pQuickDrawVars = ^QuickDrawVars;
QuickDrawVArs = RECORD
      randSeed:          LONGINT;
      screenBits:        BitMap;
      arrow:             Cursor;
      dkGray:            Pattern;
      ltGray:            Pattern;
      gray:              Pattern;
      black:             Pattern;
      white:             Pattern;
      thePort:           GrafPtr;
END; { QuickDrawVars }
```

```
FUNCTION HandleCheck (h: Handle; msgID: INTEGER): BOOLEAN;
```

`HandleCheck` checks to see if the handle `h` is NIL or empty. If it is either, `HandleCheck` returns FALSE and displays an error alert, using string `msgID` from ResEdit's 'STR#' resource ID 129. If the handle `id` is OK, `HandleCheck` returns TRUE.

```
PROCEDURE MetaKeys (VAR cmd, shift, opt: BOOLEAN);
```

`MetaKeys` returns the values of the modifier keys from the last event. Some menu commands that have shortcut key combinations simulate the shortcut modifier keys when the menu command is selected. For example, when Open Using Template is selected, `MetaKeys` indicates that the Command and Option modifier keys were pressed. Because of these transformations, `MetaKeys` should always be used to get the modifier values.

```
PROCEDURE PickEvent (VAR evt: EventRecord; pick: PickHandle);
```

`PickEvent` handles an event contained in `evt` for a standard picker referenced by `pick`. `PickEvent` should be called from your picker's `DoEvent` procedure. It is usually sufficient to call only this routine from `DoEvent`, with no other special processing at all.

```
PROCEDURE PickInfoUp (oldID, newID: INTEGER;
  pick: PickHandle);
```

`PickInfoUp` handles the update necessary when a resource's ID is changed in the Get Info window. `PickInfoUp` should be called from your picker's `DoInfoUpdate` procedure. It is usually sufficient to call only this routine from `DoInfoUpdate`, with no other special processing at all.

```
PROCEDURE PickMenu (tossOnClose: Boolean; menu, item: INTEGER;
  pick: PickHandle);
```

`PickMenu` handles menu commands for a standard picker referenced by `pick`. `PickMenu` should be called from your picker's `DoMenu` procedure. This routine handles all of the standard menu commands. If `tossOnClose` is TRUE, all of the resources displayed by the picker are released when it receives a Close command. It is usually sufficient to call only this routine from `DoMenu`.

```
FUNCTION PickStdWidth: INTEGER;
```

This function returns the width in pixels that should be used when creating picker windows. This value is obtained from the Preferences dialog box. A window of the specified width is guaranteed to fit on the screen.

```
FUNCTION  PickStdHeight: INTEGER;
```

This function returns the height in pixels that should be used when creating picker windows. This value is obtained from the Preferences dialog box. A window of the specified height is guaranteed to fit on the screen. `PickStdHeight` replaces the old `PickStdRows` procedure.

```
FUNCTION  PrintSetup: Handle;
```

Use `PrintSetup` if you are doing your own printing instead of using `PrintWindow`. Return type is actually THPrint. The following code can be used to set up your own printing loop:

```
myPrintHandle := PrintSetup;
IF myPrintHandle <> NIL THEN
   BEGIN
   PrOpen;
   IF PrError = noErr THEN
      BEGIN
      IF PrJobDialog( myPrintHandle ) THEN
         BEGIN
         printingPort := PrOpenDoc(myPrintHandle, NIL, NIL );
         IF PrError = noErr THEN
            BEGIN
{do the usual printing loop here (see TechNote #161)       }
{Warning: be careful NOT to change the current resfile  }
{               or the printing manager will fail         }
            PrCloseDoc( printingPort );
            END;
```

```
        END;
      PrClose;
      END;
    END;
```

```
PROCEDURE PrintWindow (toPrint: PicHandle);
```

`PrintWindow` does just that. If you pass it NIL, it will print an image of the current window. If you pass it a PicHandle, it will print the picture.

```
FUNCTION ResEdID: INTEGER;
```

`ResEdID` returns the resource ID of the calling picker or editor. For editors, this value should be saved in the `windowKind` field of the editor's window. For pickers, this value should be saved in the `PickId` field of the picker's `PickRec` as well as in the `windowKind` field of the window.

```
PROCEDURE SetResChanged (h: Handle);
```

`SetResChanged` sets the `resChanged` attribute for the specified resource and also sets the `mapChanged` attribute for the resource file that contains the resource. `SetResChanged` should be called whenever a resource is changed.

```
PROCEDURE SendRebuildToPickerAndFile (theType: ResType;
  parent: ParentHandle);
```

This procedure sends a rebuild (sets the rebuild flag in the window's `parentRecord`) to all open picker windows of the specified type. A rebuild is also sent to the file picker in case a new resource type is being added. This routine is useful if an editor creates a resource of another type. This routine should be called to make sure that the resource picker and the file picker are updated to reflect the addition of the new resource. For example, this routine is called from the 'ALRT', 'DLOG', and 'DITL' editors.

```
PROCEDURE SendRebuildToPicker (theType: ResType;
  parent: ParentHandle);
```

This procedure is similar to `SendRebuildToPickerAndFile` except that it doesn't send the rebuild on to the file (what a surprise!).

```
PROCEDURE SetTheCursor (whichCursor: INTEGER);
```

`SetTheCursor` changes the cursor to the specified cursor resource. The constant `arrowCursor` defined in the ResEd file should be used to set the cursor to the arrow. This routine makes sure that the resource file is set to ResEdit before loading the cursor, so that the cursor will be loaded from either ResEdit or the System file. The most common use of this routine is to set the cursor to a watch (`watchCursor`) while something is being done that may take a while.

```
PROCEDURE ShowInfo (h:Handle; dad: ParentHandle);
```

`ShowInfo` puts up a Get Info window for the resource referenced by `h` that belongs to the father object referenced by `dad`. `ShowInfo` should be called by your editor when Get Info is selected from the File menu.

```
PROCEDURE TypeToString (t: ResType; VAR s: Str255);
```

`TypeToString` returns a string consisting of the four characters that make up the `ResType` `t`.

```
PROCEDURE UseAppRes;
```

The `UseAppRes` procedure sets the current resource file to be the ResEdit Preferences file. This is necessary if you need to get a resource from ResEdit, such as a menu, string, alert, or dialog box. Be sure to restore the original resource file when you are done with ResEdit's resource file. For example:

```
SavedResFile := CurrentRes;
UseAppRes;
.
.
.
UseResFile(SavedResFile);
```

```
FUNCTION  WasItLoaded: BOOLEAN;
```

`WasItLoaded` should be called by every editor in the `EditBirth` procedure. The return value should be saved in the ParentRec data structure. When a Close command is received, the resource being edited should be released only if `WasItLoaded` returned FALSE. A return value of TRUE means the resource may already be in use by ResEdit or the System and therefore shouldn't be released.

```
PROCEDURE WritePreferences (prefType: ResType;
  prefId: INTEGER; prefName: STR255; prefHandle: Handle);
```

You can use `WritePreferences` to add your own preference resource to the ResEdit preferences file. `PrefType` is the resource type that you have chosen for your preference resource. `PrefId` and `prefName` are the ID and name for the resource. `PrefHandle` is a handle to the preference data itself. To read your preferences you can use this code:

```
myPrefs:=  Get1NamedResource(prefType, prefName);
```

Internal routines

The following routines are used internally within ResEdit and may be useful in other circumstances.

```
FUNCTION DupPick (h: Handle; c: cell; pick: PickHandle):
  Handle;
```

`DupPick` is called from `PickMenu` and should normally not need to be called from any other procedures.

```
PROCEDURE GetErrorText (error: INTEGER; VAR errorText:
  STR255);
```

`GetErrorText` will return an error string for the given error. If no specific error text is found, an I/O error is returned.

```
FUNCTION GetType (templatesOnly: BOOLEAN; VAR s: STR255):
  BOOLEAN;
```

`GetType` displays a dialog box containing a list of the types of resources that can be edited. The list contains all types for which there are templates. If `templatesOnly` is FALSE, the list also contains all the types for which there are editors. The selected type is returned in `s`. TRUE is returned if a type was selected; FALSE is returned otherwise.

```
PROCEDURE KillCache;
```

`KillCache` flushes all caches for all volumes (bitmap, control, and so on).

```
FUNCTION  MapResourceType (editor: BOOLEAN; theRes: Handle;
  origResType: ResType): ResType;
```

This function checks the 'RMAP' resources in ResEdit and the ResEdit preferences file to see if the specified resource type should be treated as if it were a different type.

```
PROCEDURE MyCalcMask (srcPtr,dstPtr: Ptr; srcRow, dstRow,
  height, words: INTEGER);
```

`MyCalcMask` calculates a mask for the given source bit image and puts it into the destination bit image. The parameters `srcPtr` and `dstPtr` reference the source and destination bit images; `srcRow`, `dstRow`, `height`, and `words` define the area on which `MyCalcMask` operates.

```
PROCEDURE NoDoubleClickHere;
```

Call this procedure in your mouse-down processing code if you don't want ResEdit to convert a doubleclick at this location to an Open command. This should be used if a double-click makes sense only in part of your window.

```
FUNCTION  PlaySyncSound(which: INTEGER; sndHandle: Handle):
  BOOLEAN;
```

`PlaySyncSound` is used by the 'snd ' picker to play sounds.

```
FUNCTION ResEditRes: INTEGER;
```

The `ResEditRes` procedure returns the resource file ID of ResEdit. This routine will rarely be needed. You can use this routine if you don't want to release a resource that you have been editing, if the resource came from ResEdit.

```
FUNCTION  RestoreRemovedResources (pick: PickHandle): BOOLEAN;
```

This function reverts all resources of the type handled by the picker (`pick^^.rType`). It returns true if the list needs to be rebuilt.

```
PROCEDURE ScrapCopy ( VAR h: Handle );
```

`ScrapCopy` copies the handle `h` into the ResEdit scrap. A different handle will be returned.

```
PROCEDURE ScrapEmpty;
```

`ScrapEmpty` empties the ResEdit and desktop scrap.

```
PROCEDURE ScrapPaste (pasteAll: BOOLEAN; typeToPaste: ResType;
  resFile: INTEGER);
```

`ScrapPaste` pastes the resources from the ResEdit scrap to the file identified by the ID number `resFile`. If `pasteAll` is TRUE, all resources found in the scrap are pasted. If `pasteAll` is FALSE, only resources of type `typeToPaste` are pasted.

The next four routines implement the color palette pop-up menu used by the 'MENU' editor.

```
PROCEDURE InstallColorPalettePopup( whichWindow: WindowPtr;
  CQDishere, isActive: Boolean );
```

`InstallColorPalettePopup` sets up a palette for the window containing the system colors for the deepest available device. Call this procedure immediately after opening your window and whenever you receive an update event. `whichWindow` is the window containing the pop-up menu, `CQDishere` is TRUE when Color QuickDraw is available, and `isActive` is TRUE when the window is the frontmost one.

```
PROCEDURE DrawColorPopup( whichWindow: WindowPtr;
  itemBox: Rect; whichColor: RGBColor;
  CQDishere: Boolean );
```

DrawColorPopup draws the color patch and a drop shadow indicating that this is actually a pop-up menu. Call this procedure for every pop-up palette whenever you need to update the window contents. `whichWindow` is the window containing the pop-up palette, `itemBox` is the `Rect` to be used to draw the color patch, `whichColor` is the RGBColor to be drawn and `CQDishere` is TRUE when Color QuickDraw is available.

```
FUNCTION ColorPalettePopupSelect( whichWindow: WindowPtr;
  itemBox: Rect; VAR whichColor: RGBColor;
  CQDishere: Boolean ): Boolean;
```

`ColorPalettePopupSelect` handles mouse-down events in the color palette pop-up menu. Call this procedure whenever you receive a mouse-down event in one of your color patches. `whichWindow` is the window containing the pop-up, `itemBox` is the `Rect` to be used to draw the color patch, `whichColor` is the RGBColor to be used as default and `CQDishere` is TRUE when Color QuickDraw is available. On exit, `whichColor` contains the RGBColor selected by the user.

```
PROCEDURE DeinstallColorPalettePopup( whichWindow: WindowPtr;
  CQDishere: Boolean );
```

`DeinstallColorPalettePopup` removes the palette from the window. Call this procedure before closing the window.

Obsolete routines

The following routines are obsolete and should no longer be used. They are no longer available in the current version of ResEdit.

```
FUNCTION CWindSetup (width, height: INTEGER; t, s: STR255):
  WindowPtr;
```

Use `PickerWindSetup` or `EditorWindSetup` instead.

```
FUNCTION WindSetup (width, height: INTEGER; myType, name:
  STR255): WindowPtr
```

Use `PickerWindSetup` or `EditorWindSetup` instead

```
FUNCTION PickStdRows: INTEGER;
```

No longer supported. Use `PickStdHeight` instead.

```
PROCEDURE CallPBirth (theType: ResType; parent: ParentHandle;
  id: INTEGER );

PROCEDURE CallEBirth (resHandle: Handle; parent: ParentHandle;
  id: INTEGER );

PROCEDURE CallEvent( VAR evt: EventRecord; refcon: LONGINT;
  id: INTEGER );

PROCEDURE CallMenu (menu, item: INTEGER; refcon: LONGINT;
  id: INTEGER);

FUNCTION CopyRes (VAR h: Handle; makeID: BOOLEAN;
  resNew: INTEGER): Handle;

PROCEDURE DoKeyScan (var evt: EventRecord; offset: integer;
  lh: ListHandle);

PROCEDURE DoListEvt (e: EventRecord; l: ListHandle);
```

Appendix A The 'KCHR' Resource

This appendix contains more information about the 'KCHR' resource, its structure, and its function. The 'KCHR' resource controls mapping from the keyboard to the resulting characters. This mapping process involves several areas of the Macintosh architecture.

Basic theory of keyboard operation

In order to appreciate fully the workings of the 'KCHR' editor, you really should be aware of the process that it controls. Here is a summary.

Generating the virtual keycode

Whenever a key on any type of keyboard is pressed, the operating system polls the key information from the device. It then translates each raw keycode generated by the keyboard into a virtual keycode and a combination of modifier keys by means of the 'KMAP' resource. The resulting virtual keycode is information about the key being depressed that is independent of the keyboard type.

Exceptions to the rule

Some countries have different layouts for different keyboards, mostly for historical reasons. To deal with those exceptions, the 'itlk' resource contains a table of translation rules from a virtual keycode generated by the actually connected keyboard to a virtual keycode on the ISO ADB keyboard or to whatever keyboard is supported by the 'KCHR' resource for that country.

Generating the character code

When the operating system has generated a virtual keycode, the `KeyTrans()` procedure then translates the virtual keycode and the concurrently pressed modifier keys into a Macintosh character set number based on the tables in the 'KCHR' resource. That character number and the virtual keycode information are then stored in the event queue and can be accessed by calling `GetNextEvent()`.

Dead keys

When you press a dead key, the first thing you'll notice is that nothing happens immediately (that is, no event is fed into the queue). When you then press another key, the Event Manager uses the character number generated by this new key and the previously pressed dead key to determine which character number should be put in the event queue. This process is used, for example, to generate the German characters with umlauts Ä,Ö,Ü,ä,ö, and ü. You have to press the dead key for a diaeresis (which is Option-u in the U.S. 'KCHR') and then press one of the keys that generate the characters A,O,U,a,o, or u. (You can also generate ï, and ë, which do not exist in German, but, depending on the font, possibly not their uppercase equivalents.) If you press a key that generates none of the defined character numbers for this dead key, the Event Manager generates the nomatch character (which is, in the case discussed here, the umlaut alone).

The Dead Array contains a list of dead keys. For each dead key it defines the virtual keycode and the table that is used to trigger the dead-key mechanism. It then lists pairs of completion characters and substitution characters and, finally, the nomatch characters. The whole dead-key mechanism can be described as follows:

1. Press a dead key on the keyboard.
2. Press any key that generates a character number that corresponds to a valid completion character.

You get the corresponding substitution character in the event queue. (If you didn't press a valid completion character in step 2, you get the nomatch character.)

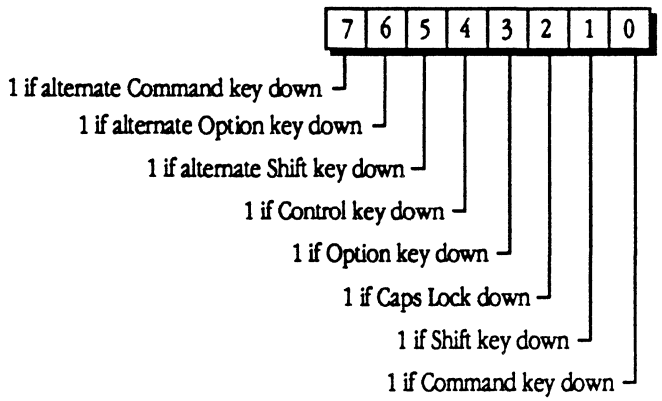
The structure of a 'KCHR' resource

Here is the definition of a 'KCHR' for the resource compiler Rez. (This information can also be found in the file SysTypes.r in the folder {RIncludes} in MPW.)

```
type 'KCHR' {
  integer;                                  /* Version                */
  wide array [$100] {                       /* Indexes        */
    byte;
  };
  integer = $$CountOf(TableArray);
  array TableArray {
    wide array [$80] {    /* ASCII characters                */
      char;
    };
  };
  integer = $$CountOf(DeadArray);
  array DeadArray {
    byte;         /* Table number       */
    byte;         /* Virtual keycode    */
    integer = $$CountOf(CompletorArray);
    wide array CompletorArray {
      char;       /* Completing char          */
      char;       /* Substituting char        */
    };
    char;         /* No match char1           */
    char;         /* No match char2           */
  };
};
```

Each table in the Table Array describes the virtual keycode-to-character number translation for one complete layer of the keyboard (that is, for all 128 possible keys). The Index Array defines the mapping of modifier key combinations to tables. The high byte of the modifier flag (described in *Inside Macintosh*, Volume V, Chapter 10) is used as an index to determine the number of the table to be used for translation. The information in *Inside Macintosh* is, however, not complete, because the alternate modifier keys (the Shift, Option, and Control keys on the right side of the ADB extended keyboard) are not mentioned. Those keys are normally coupled with the corresponding keys on the left side. It is possible to uncouple them by sending a command to the keyboard. (See "Reassigning Right Key Code" in *Inside Macintosh*, Volume V, Chapter 10.) The correct bit layout of the high byte is shown in Figure A-1.

■ **Figure A-1** Modifier flag high byte



Suppose you hold down the Option key. This keypress will result in a value of 8 (bit 3 is set) in the high byte of the modifier flag. Thus the Toolbox Event Manager takes the value stored in `IndexArray[8]`, which is 3 in the current U.S. 'KCHR', and therefore uses Table 3 to translate the keycodes to character numbers.

Appendix B The 'BNDL' Resource

The 'BNDL' resource bundles together icons (resource types 'ICN#', 'ics#', 'icl4', 'icl8', 'ics4', 'ics8'), file type references (resource type 'FREF'), and the "signature" resource (whose resource type is identical to the creator field of the application file) for the Finder. This enables the Finder to display distinct icons for an application and its documents, and also enables it to launch the appropriate application when the user double-clicks a document.

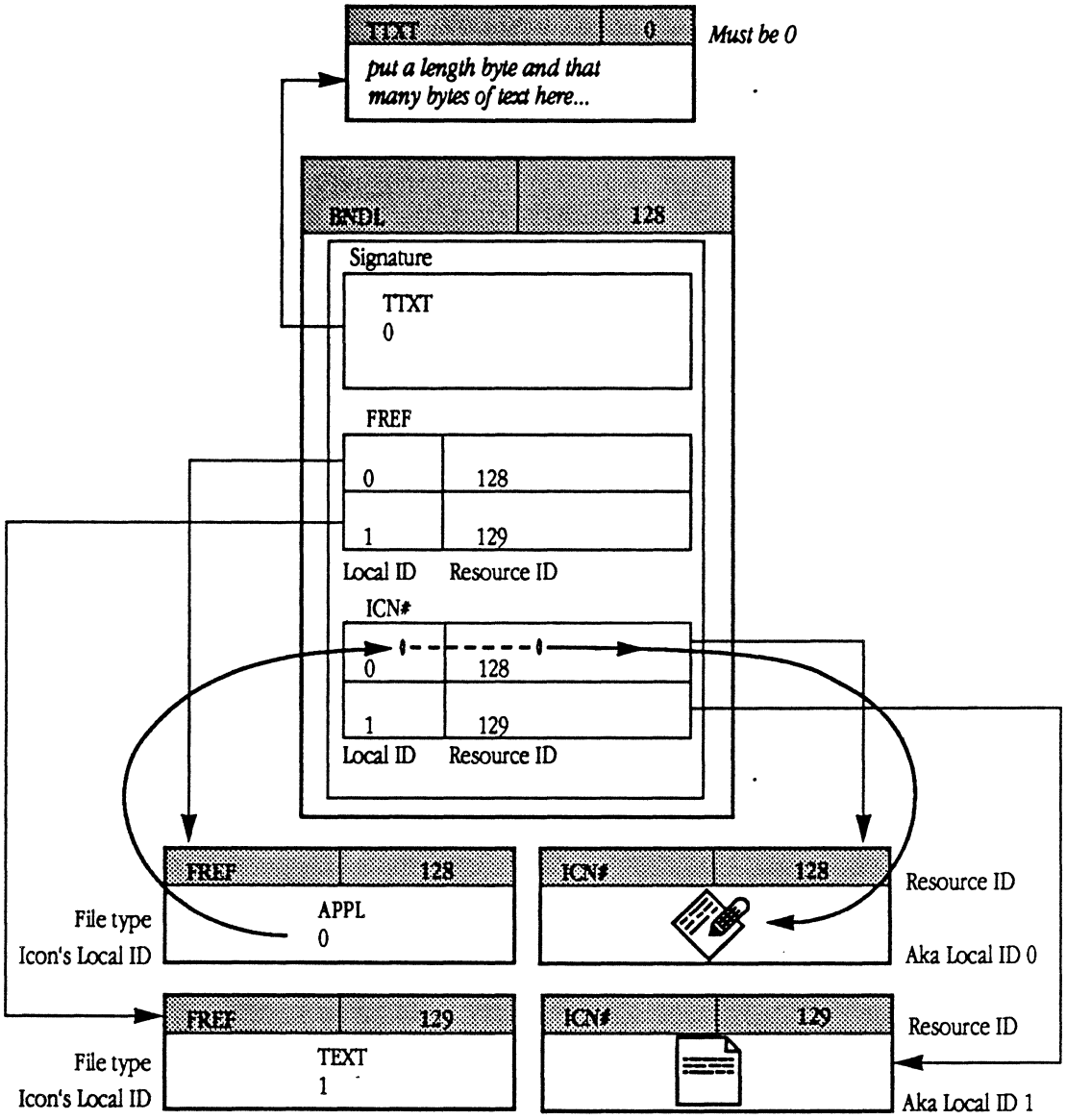
The structure of a 'BNDL' resource

The 'BNDL' resource contains a reference to the signature resource type and ID (for historical reasons the ID must be 0) as well as a list of resource types (almost always only 'FREF' and 'ICN#', although other things are theoretically possible) and localID to resourceID mapping tables. The term "local" ID is used, because this ID is used within the 'BNDL' resource itself to tie together the file reference and its icons. When the Finder copies the 'BNDL' resource and all its bundled resources to the Desktop file (or the desktop database in System 7.0), it actually has to change the resource ID numbers to avoid ID conflicts within the Desktop. The local ID numbers remain unchanged.

The signature resource can contain anything you want, although, for historical reasons, it typically contains some version and copyright information. The resource ID of the signature resource needs to be 0. If you use the 'BNDL' editor in ResEdit 2.0 this resource is transparently created and maintained for you.

For every file type that should be displayed with a distinct icon in the Finder there need to be two entries in the 'BNDL' resource, which in turn refer to one 'FREF' resource, and one 'ICN#' resource (or an entire Finder icon family for system software version 7.0). The 'FREF' resource contains the 4 character file type and a reference to a local ID for an icon to be used for this file type. Even if you plan to include an entire Icon Family, you only need to list the 'ICN#' resource in the 'BNDL' resource. The System 7.0 Finder automatically recognizes and loads all the other parts of the Finder Icon Family. The relationship of local ID numbers and resource ID numbers is shown in Figure B-1.

■ **Figure B-1** Six resources and their relationships



In order for the Finder to recognize a 'BNDL' resource these conditions must be met:

1) The bundle must be complete; that is, all the resources listed here must exist and their relationships must be defined. If you use the 'BNDL' resource editor built into ResEdit 2.0 you can be sure that this condition is met.

2) The file's creator must be identical to the signature specified in the 'BNDL' resource and the file's file type must be one listed in the 'BNDL' (i.e., it must have its own 'FREF' and corresponding 'ICN#'). Typically the file type will be 'APPL' for application, although any file can contain 'BNDL' resources. Specific examples other than 'APPL' are 'INIT' and 'CDEV'. Use the Get File/Folder Info command in the File menu to change the file's file type or creator.

3) The file's Bundle bit must be set and the Inited bit must be cleared. The Finder always sets the Inited bit whenever it finds a new file and reads in some information about it. By clearing this bit you tell the Finder to reread that information. Use the Get File/Folder Info command in the File menu to change the Bundle and Inited bits.

4) There must not already be a 'BNDL' resource with the same signature in the Desktop file (or desktop database in System 7.0). If you want to change an existing bundle (to modify the icons, for example) you will need to recreate the Desktop file by rebooting while holding down the Option and Command keys. Note that by doing so you will lose all comments you may have entered in the Get Info windows in the Finder in system software before version 7.0.

Definitions of the 'BNDL' and 'FREF' resources

Here are the definitions of the 'BNDL' and 'FREF' resources from the MPW Types.R file:

```
/*-------------------------BNDL • Bundle-------------------------------*/
type 'BNDL' {
            literal longint;
                  /* Signature                    */
            integer;
                        /* Version ID                   */
            integer = $$CountOf(TypeArray) - 1;
            array TypeArray {
                        literal longint;
                  /* Type                         */
                        integer = $$CountOf(IDArray) - 1;
                        wide array IDArray {
                                    integer;
                        /* Local ID                     */
                                    integer;
                        /* Actual ID                    */
                        };
            };
```

```
};
/*-------------------FREF • File Reference-----------------------------*/
type 'FREF' {
            literal longint;
                  /* File Type                  */
            integer;
                        /* Icon ID                        */
            pstring;
                        /* Filename                       */
};
```

Appendix C Resource Types Defined for Rez and ResEdit

This appendix contains a list of some resource types in use at Apple Computer, Inc., current as of mid-1990. An attempt has been made to give pertinent information about what each type is, how it is handled by the resource compiler, Rez, and how it is handled by ResEdit. This list is neither formal nor exhaustive!

■ **Table C-1** Resource types defined for Rez and ResEdit

Type	Definition	Rez	ResEdit
actb	Alert Color Lookup Table	Types.r	Template
acur	Animated cursor resource	Types.r	Template
ADBS	ADB driver loaded before INIT 31	-----	-----
ALRT	Alert Template	Types.r	Template, Editor
APPL	Application list (Desktop)	-----	Template
atpl	AppleTalk Resource	-----	-----
bmap	BitMap	-----	-----
BNDL	Bundle	Types.r	Template, Editor
CACH	RAM Cache Control Code	-----	-----
cctb	Control Color Lookup Table	Types.r	Template
CDEF	Code for drawing controls	-----	-----
cicn	Color Icon	Types.r	-----, Editor
clut	Generic Color Lookup Table	Types.r	Template
CMDO	For MPW commando interface	Cmdo.r	-----
cmnu	MacApp temporary menu resource	-----	-----, Editor
CNTL	Control Template	Types.r	Template
CODE 0	Jump Table	-----	-----
CODE	Application Code	-----	-----
crsr	Color Cursor	Types.r	-----
ctab	Cache Tab <list of possible cache sizes>	-----	-----
CTY#	City list from MAP CDEV	-----	Template
CURS	Cursor	Types.r	-----, Editor
dctb	Dialog Color Lookup Table	Types.r	Template
DICL	<for MacWorkstation>	-----	-----
DITL	Dialog Item List	Types.r	Template, Editor
DLOG	Dialog Template	Types.r	Template, Editor
DRVR	Driver	SysTypes.r	Template
DSAT	Startup alerts & code to display them	-----	-----
FBTN	MiniFinder button	-----	Template
fctb	Font Color Lookup Table	Types.r	Template

(Continued)

■ **Table C-1** Resource types defined for Rez and ResEdit (continued)

Type	Definition	Rez	ResEdit
FCMT	GetInfo comments from Desktop file		
		-----	Template
FDIR	MiniFinder button directory ID	-----	Template
finf	Font information	SysTypes.r	Template
FKEY	Function Key Code	-----	-----
fld#	List of folder names for folder msg		
		SysTypes.r	Template
FMTR	Format Record	-----	-----
FOBJ	Information about Folders	-----	-----
FOND	Font Family Description	SysTypes.r	Template
FONT	Font Description	SysTypes.r	Template, Editor
FREF	File Reference	Types.r	Template
FRSV	ROM Font resources	-----	Template
FWID	Font Width Table	SysTypes.r	Template
gama	Gamma Table (color correction for screen)		
		-----	-----
GNRL	NBP Timeout and retry info for AppleTalk		
		-----	-----
ICON	Icon	Types.r	-----, Editor
ICN#	Icon List	Types.r	-----, Editor
ictb	Color dialog item list	-----	-----
INIT	Code that is run at system startup time		
		-----	-----
insc	Installer Script	SysTypes.r	Template
INTL 0	International Formatting Information		
		SysTypes.r	-----, Editor
	(== itl0; no longer used)		
itl0	International Formatting Information		
		SysTypes.r	-----, Editor
INTL 1	International Date/Time Information		
		SysTypes.r	-----, Editor
	(== itl1; no longer used)		

(Continued)

■ **Table C-1** Resource types defined for Rez and ResEdit (continued)

Type	Definition	Rez	ResEdit
itl1	International Date/Time Information		
		SysTypes.r	-----, Editor
itl2	Intl Str Comparison Package Hooks		
		SysTypes.r	-----
itl4	International Tokenize	SysTypes.r	-----
itlb	International Script Bundle	SysTypes.r	-----
itlc	International Configuration	SysTypes.r	-----
itlk	Intl exception dictionary for kchar		
		SysTypes.r	Template
KCAP	Physical Layout of Keyboard	SysTypes.r	Template
KCHR	ASCII Mapping (software)	SysTypes.r	-----, Editor
KEYC	old keyboard layout <used by old INIT 0+1>		
		-----	-----
KMAP	Keyboard Mapping (hardware)	SysTypes.r	Template
kscn	Keyboard/Script icon	Types.r	-----
KSWP	Keyboard Swapping	SysTypes.r	Template
LAYO	Finder layout resource	-----	Template
LDEF	Code for drawing lists	-----	-----
mach	cdev filtering	SysTypes.r	-----
MACS	Version # in system file	-----	Template
MBAR	Menu Bar	Types.r	Template
MBDF	Menu bar definition procedure <Code>		
		-----	-----
mcky	Mouse Tracking	SysTypes.r	Template
mctb	Menu Color Lookup Table	Types.r	-----, Editor
mcod	MacroMaker information	-----	-----
mdct	MacroMaker information	-----	-----
MDEF	Code for drawing menus	-----	-----
mem!	MacApp memory utilization	-----	-----
MENU	Menu	Types.r	Template, Editor
minf	Macro info (MacroMaker)	-----	Template
mitq	Default queue sizes for MakeITable		
		SysTypes.r	-----

(Continued)

■ **Table C-1** Resource types defined for Rez and ResEdit (continued)

Type	Definition	Rez	ResEdit
mntb	MacApp menu table(relate command # to menu)		
		-----	-----
mppc	MPP Configuration Resource	SysTypes.r	-----
NBPC	NBP configuration <AppleTalk>	-----	-----
ncts	List of constants	-----	-----
NFNT	Font Description	SysTypes.r	-----
nrct	Rectangle position list	SysTypes.r	Template
PACK	Packages of code used as ROM extensions		
		-----	-----
PAPA	Printer Access Protocol Address (AppleTalk)		
		-----	Template
PAT	Quickdraw Pattern	Types.r	-----, Editor
PAT#	Quickdraw Pattern List	Types.r	-----, Editor
PDEF	Code to drive printers	-----	-----
PICT	Quickdraw Picture	Types.r	Template
pltt	Color Palette	Types.r	Template
POST	Postscript - found in Laser Prep file		
		-----	Template
ppat	Pixel Pattern	Types.r	Template
ppt#	Array of ppats	-----	-----
PREC	Printer driver's private data storage		
		-----	-----
PRC0	Default page setup info for printer		
	(PRED id = 0)	-----	Template
PRC3	Print record (PREC id = 3)	-----	Template
PSAP	Just a string	-----	Template
PTCH	ROM Patch	-----	-----
qrsc	System 7.0 query resource	-----	Template
ROv#	ROM Resource Override	SysTypes.r	Template
scrn	Screen Configuration	SysTypes.r	Template
seg!	MacApp	-----	-----
SERD	RAM serial driver	-----	-----
SICN	Small Icon	Types.r	-----, Editor
SIGN		-----	Template

(Continued)

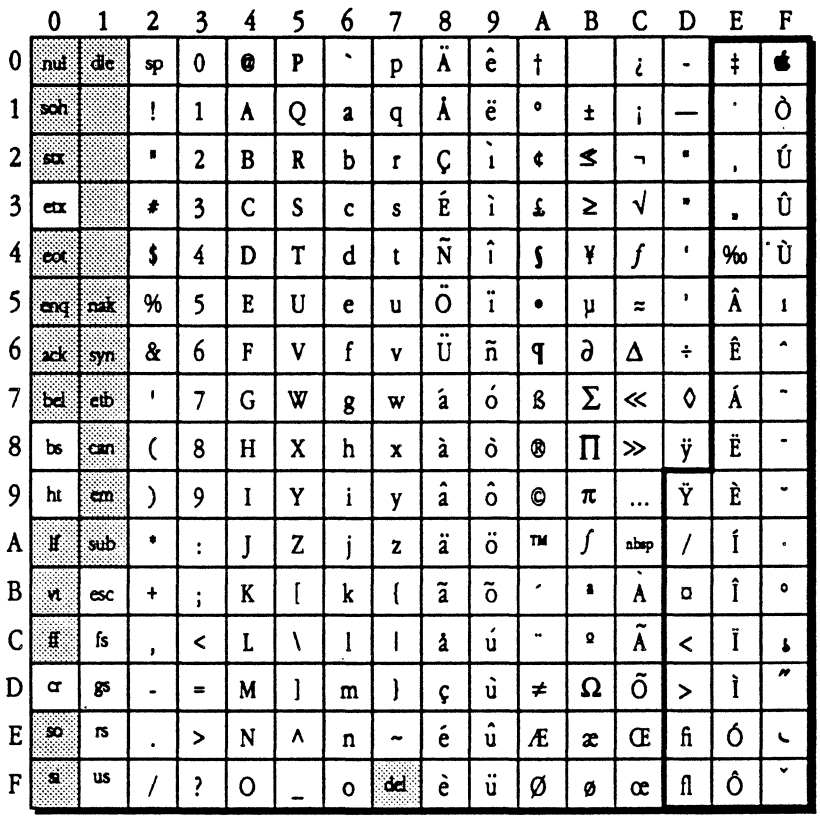
■ **Table C-1** Resource types defined for Rez and ResEdit (continued)

Type	Definition	Rez	ResEdit
SIZE	MultiFinder Size Information	Types.r	Template
snd	Sound	SysTypes.r	----- (player)
STR	PascalStyle String	Types.r	Template
STR#	PascalStyle String List	Types.r	Template
styl	Style information for TextEdit	-----	-----, Editor
TEXT	Unlabeled string.(Same as minf)	-----	Template
tlst	Title list	-----	-----
TMPL	ResEdit template	-----	Template
vers	Version	SysTypes.r	Template
wctb	Window Color Lookup Table	Types.r	Template
WDEF	Code for drawing windows	-----	-----
WIND	Window Template	Types.r	Template, Editor
wstr	Query str used by qrsc resource	-----	Template

Appendix D **The Macintosh Character Set**

This appendix contains a chart that displays the regular character set for Macintosh fonts. The first 128 characters correspond to the standard ASCII character set. Please remember that not all fonts for the Macintosh have these standard characters in them. Specific examples are Symbol and ITC Zapf Dingbats; there are also many pictorial fonts available as bitmaps for dot-matrix printing.

■ **Figure D-1** Macintosh character set



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	nul	dle	sp	0	@	P	`	p	Ä	ê	†		¿	-	‡	
1	soh		!	1	A	Q	a	q	Å	ë	°	±	¡	—	·	Ò
2	stx		"	2	B	R	b	r	Ç	ì	¢	≤	¬	"	‚	Ú
3	etx		#	3	C	S	c	s	É	í	£	≥	√	"	„	Û
4	eot		$	4	D	T	d	t	Ñ	î	§	¥	ƒ	'	‰	Ù
5	enq	nak	%	5	E	U	e	u	Ö	ï	•	µ	≈	'	Â	ı
6	ack	syn	&	6	F	V	f	v	Ü	ñ	¶	∂	Δ	÷	Ê	^
7	bel	etb	'	7	G	W	g	w	á	ó	ß	Σ	«	◊	Á	~
8	bs	can	(	8	H	X	h	x	à	ò	®	∏	»	ÿ	Ë	¯
9	ht	em	)	9	I	Y	i	y	â	ô	©	π	…	Ÿ	È	˘
A	lf	sub	*	:	J	Z	j	z	ä	ö	™	∫	nbsp	/	Í	˙
B	vt	esc	+	;	K	[	k	{	ã	õ	´	ª	À	¤	Î	°
C	ff	fs	,	<	L	\	l	|	å	ú	¨	º	Ã	<	Ï	¸
D	cr	gs	-	=	M	]	m	}	ç	ù	≠	Ω	Õ	>	Ì	˝
E	so	rs	.	>	N	^	n	~	é	û	Æ	æ	Œ	fi	Ó	˛
F	si	us	/	?	O	_	o	del	è	ü	Ø	ø	œ	fl	Ô	ˇ

sp space

del delete —

nbsp non-breaking space (option-space on US keyboard)
The key labeled Delete on the US keyboard actually generates bs (08) character.

The shaded characters cannot normally be generated from the Macintosh keyboard or keypad.

Index

F

G

I, J

U

V

W

X, Y, Z

THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh® computers and Microsoft® Word software. Proof and final pages were created on Apple LaserWriter® printers. Line art was created using Adobe Illustrator™. POSTSCRIPT®, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated. Screen shots were taken with FlashIt.

Text type and display type are Apple's corporate font, a condensed version of Garamond. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.

Writer: Jon Singer

Developmental Editor: Silvio Orsino and Steve Hiatt
Illustrator: Deb Dennis and Sandee Karr
Production Supervisor: Renee Ekleberry

Special thanks to:

Nobu Toge for FlashIt.

Mikel Evins for DreadEdit.

The ResEdit engineering team, particularly Peter, Craig, and Alexander, who helped the author more than he can say.

Developer Technical Support at Apple for assistance above and beyond the call of nature, and for Clarus the DogCow. Moof!™