 . Macintosh®

Macintosh® ResEdit 1.2 Reference

Final Draft.

Caution! This book is NOT a release version! It is for the Apple Computer Technical Library only! Do not distribute this book, or we will find you, and you won't like it!

APPLE COMPUTER, INC.

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

Simultaneously published in the United States and Canada.

2/23/89

Table of Contents

Figures and Tables

Chapter 1 **ResEdit Overview**

THIS CHAPTER INTRODUCES RESEDIT™, a stand-alone application for editing resources. ResEdit is an interactive, graphically based application for manipulating the various resources in a Macintosh® file. (Some Macintosh files don't have any resources, but all applications and most of the System Folder files do.) If you are used to other computers, you will rapidly discover that resources are handled very differently on the Macintosh computer. ■

Resources

One of the features that distinguishes the Macintosh from other computers is the way it handles what we call *resources* (fonts, sounds, icons, patterns, and so on). In most computers, a "file" consists of a set of bytes, perhaps beginning with a header that contains information about what kind of file it is, and where various pieces are to be found inside the file. Fonts and other resources, on the other hand, tend to be kept in a central pool, which is typically part of the operating system, and can be reached only by accessing that pool. The Macintosh has, instead, a file structure that can have two sets of bytes (a resource fork and a data fork), so that resources may be housed within any file, in an area set aside specifically for that purpose. Any file can have a resource fork, and any file can have a data fork. Some have only one, some have both. For example, a typical MacWrite® document has only a data fork, containing text and its associated formatting information; but there is no rule to forbid you from changing the situation. You could, for example, build a font, write some documentation for it, and ship font and documentation together by giving the documentation file a resource fork and putting the font into it.

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 prime 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 ').

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 necessary resources, such as menus. Apple reserves all names that don't contain any uppercase letters. Anything with at least one uppercase letter in it is yours to use, though it is a good idea to check to be sure that you don't accidentally use a type name that is already in use by someone else. The Developer Technical Support group at Apple Computer, Inc. maintains a registry of resource types for registered developers. 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.

ResEdit lets you copy and paste all resource types, and lets you edit many of them. ('NFNT' is a notable 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.

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. There is information on existing templates and on generating your own templates in Chapters 4 and 5, and an example of template editing in Chapter 6.

Resources of the third kind are edited with the general (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 in the process of putting together a quick prototype of a user interface, to try out different formats and presentations of resources. 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' 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 'BNDL' and 'FREF' resources, for example. The layout of these dialog boxes is determined by a template in ResEdit's resource file, and you can add templates to edit new resource types. Resource templates are described in Chapters 4 and 5.
- You can program your own special-purpose **resource picker** or **editor** (or both), and add it to ResEdit. (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 8 for information.

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 MPW's 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 editor or another text editor. Rez and DeRez are fully described in the *Macintosh Programmers 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 Rez 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. It is a good idea to edit spare copies of files rather than originals, and to avoid editing the contents of the System Folder on the current startup volume. (Under MultiFinder™, you cannot use ResEdit to open the current Finder or Desktop file.) Remember, also, that if you open a file that is already open (for example, the currently active System file, or ResEdit itself), and you make any changes, you *must* save those changes unless you revert them before you attempt to close the file! ■

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 ...
```

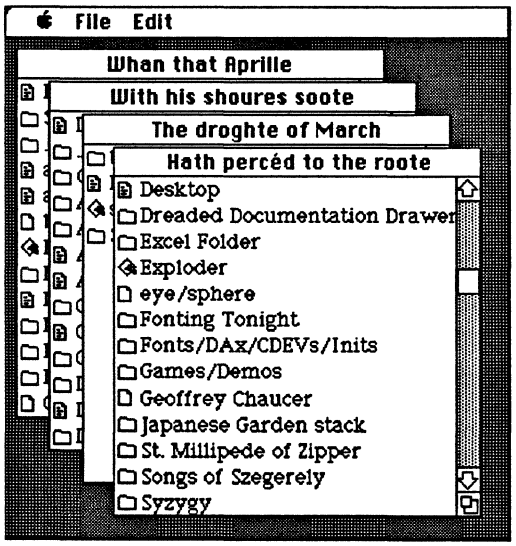
The latter command causes ResEdit to open the named files automatically.

ResEdit displays a window for each disk volume currently mounted. (See Figure 2-1.) Each window shows a list of the files and folders available at the top directory level of that volume. There is a close box in the upper-left corner of the window of a removable volume (a floppy disk, for example). If you click the close box, ResEdit removes all windows associated with the volume, and unmounts the volume. (If it is a floppy disk, it is ejected.) Nonremovable volumes, such as hard disk drives, do not have close boxes, and cannot be unmounted.

If a volume is mounted during a ResEdit session, a window for that volume is created. If a volume is unmounted during a session, all windows associated with that volume are closed.

Notice that disk windows (as well as many other windows in ResEdit) are resizeable.

■ **Figure 2-1** Disk volume windows



Working with files

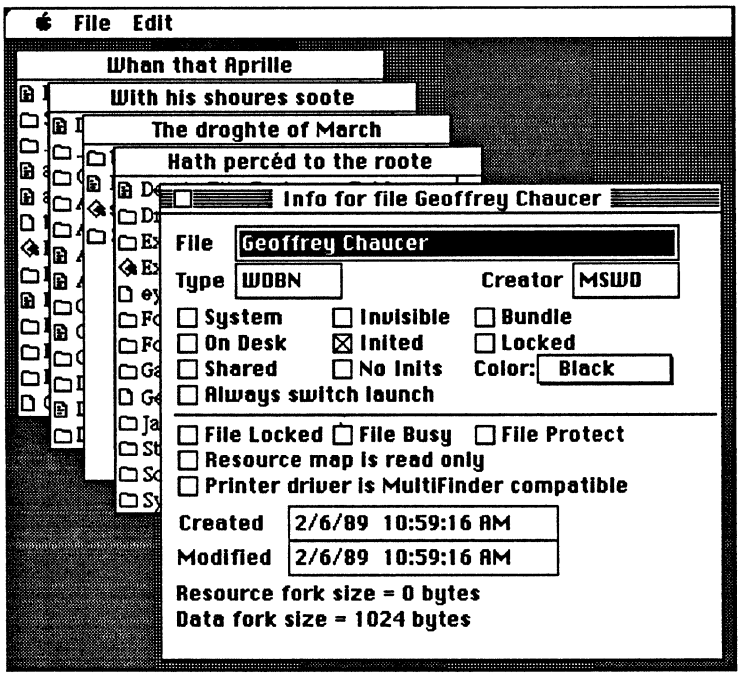
You can select a filename by clicking it or by typing one or more characters of the filename. To select more than one item, hold down the Command key while clicking the individual items; or click an item at the beginning of the range you want to select, hold down the Shift key, and click the item at the end of the range. You can, of course, then continue to select or deselect individual items with the Command key. (These techniques will also work at other levels within ResEdit, for example to select resource types and, within a type, to select individual resources.) To list the resource types in a file, select and open the filename from the list. If you try to open a file that does not have a resource fork, ResEdit displays a dialog box that asks you whether you want to open the file anyway. If you say yes, the file is opened and given a resource fork.

When a directory window is the active window, the File menu commands act as follows:

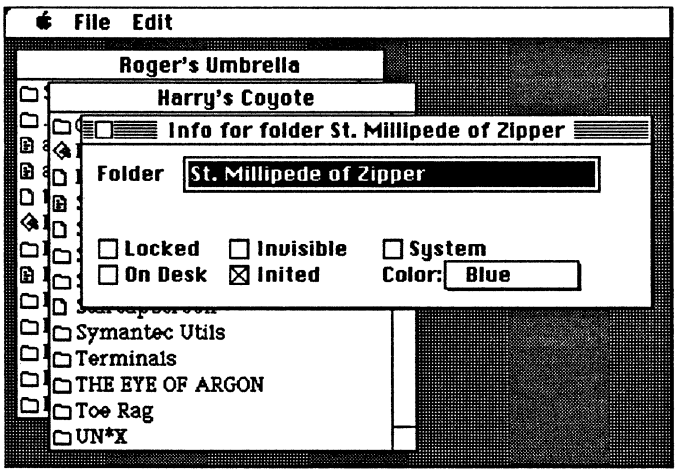
New	Creates a new file.
Open	Opens the selected file or folder. (Choosing this command has the same effect as double-clicking the filename or selecting the filename and pressing the Return key or the Enter key.)
Close	Closes the volume window. (Using this command has the same effect as clicking the close box.) If the volume is removable, it is removed. (A floppy disk, for example, is ejected.)
Save	Not usable at this level.
Get Info	Displays file or folder information and allows you to change it. Figure 2-2 is an example of a File Info window. Figure 2-3 is an example of a Folder Info window.
Transfer...	Allows you to transfer to an application other than the application that launched ResEdit, without first returning to the Finder.
Quit	Quits ResEdit and returns to the Finder (or the MPW Shell, HyperCard, or whatever program launched ResEdit).

▲ Warning You can edit any file shown in the window, including the System file and ResEdit itself (though there are some restrictions under MultiFinder). It's dangerous, though, to edit a file that's currently in use, and it's important to remember that if you must do so, and if you make any changes to the file, you must save those changes. In general, it is much wiser to edit a copy of the file instead. ▲

■ **Figure 2-2** A File Info window



■ **Figure 2-3** A Folder Info window



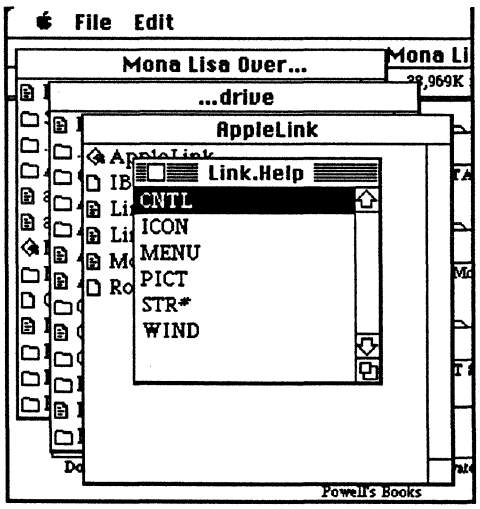
ResEdit recognizes a new disk when it's inserted, and handles multiple disk drives. Note that you can also use ResEdit to delete files:

- To delete a file, select the file and choose Clear from the Edit menu.
- To copy a resource file, select all of its resources and copy them. Then paste them into a new file. (File attributes are not automatically copied by this operation—you must set them via the Get Info command.) *ResEdit cannot copy a data fork.*

Working within a file

When you open a file, a **file window** appears. This window displays a list of all the resource types in that file (Figure 2-4). While this window is the active window, you can create new resource types, copy or delete existing resources, and paste resources from other files. At this level, all 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.

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



When a file window is the active window, the File menu commands have the following effects:

New	Creates a new resource type in the open file. If a resource of the specified type already exists, its **file type window** is opened. This command does not create any resources when used at the file level.
Open	Opens a file type window that displays all resources of the resource type selected. (Select the resource type by clicking it or by typing its first character, if that's unique, or first few characters.)

◆ *Note:* If you hold down the Option key while opening a resource type, the resource window will open with the **general resource picker**. This procedure is equivalent to choosing Open General, described next. Resource pickers are explained in "Working Within a Resource Type," later in this chapter.

Open general	Opens a window displaying all resources of the selected type, using the general resource picker.
Close	Closes all windows that are open for this file and asks if you want to save the changes you have made.
Save	Saves the changes you have made.
Revert	Changes the resource file back to the version that was last saved to disk.
Transfer...	Quits ResEdit, allowing you to transfer control to another application.
Quit	Quits ResEdit. If you have made any changes, ResEdit asks whether you want to save them.

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

Undo	Not usable at the file level.
Cut	Removes all resources of the resource types selected, placing them in the ResEdit scrap.
Copy	Copies all resources of the resource types selected into the ResEdit scrap. Copy puts a single copied resource into the Clipboard. If you copy multiple resources, ResEdit clears the clipboard and puts the resources into its own scrap.

Paste	Copies the resources from the ResEdit scrap (or from the clipboard, if only one resource is involved) into the file window's resource type list.

- *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®, both an 'MDPL' and a 'PICT' are put into the scrap. When you paste into the file window in ResEdit, all resources that are present will be pasted.

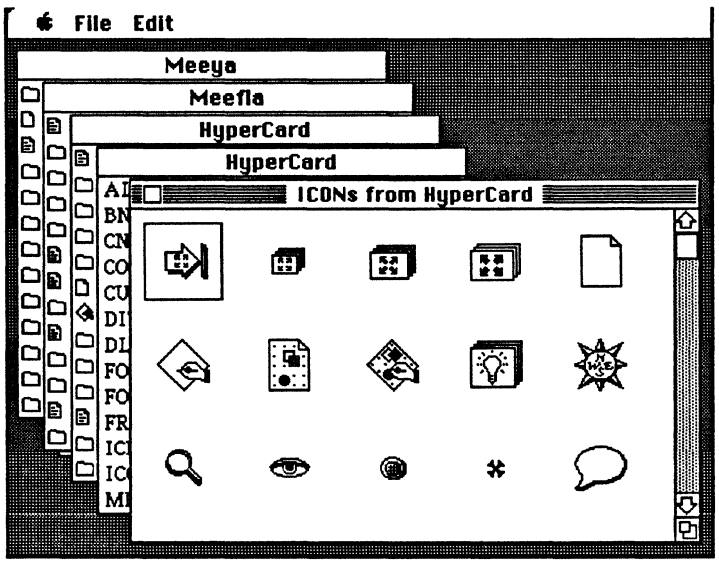
Clear	Removes all resources of the resource type selected, without placing them in the ResEdit scrap.
Duplicate	Not usable at the file level.

Working within a resource type

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. If you hold down the Option key during the open or if you select Open General from the File menu, the general resource picker is invoked even if there is a picker specifically configured for the resource type you are opening. The general resource picker displays the resources by type, name, and ID number; pickers for specific resource types generate displays that are appropriate for their type. Figure 2-5 shows a picker for the 'ICON' resource type.

You can also write your own pickers. For more information, see Chapter 8.

■ **Figure 2-5** A resource type window (with custom picker)



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

New — Creates a new resource and opens its editor.

Open — Opens the appropriate editor for the resource you selected.

Open as Template... — Lets you open a resource using a template you specify.

◆ *Note:* If you hold down the Option and Command keys while opening a resource, the effect is the same as that of choosing Open as Template.

Open General — Opens the general resource editor.

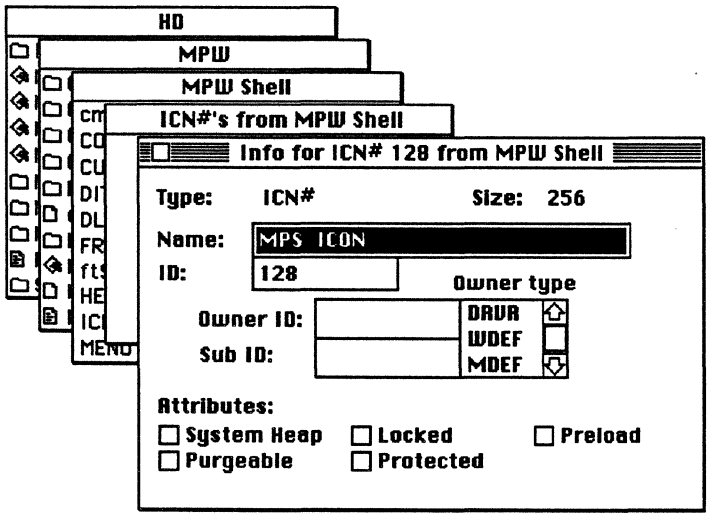
Close — Closes the resource type window and any editor windows that exist for resources of the current type.

Save — Saves the changes you have made to the file.

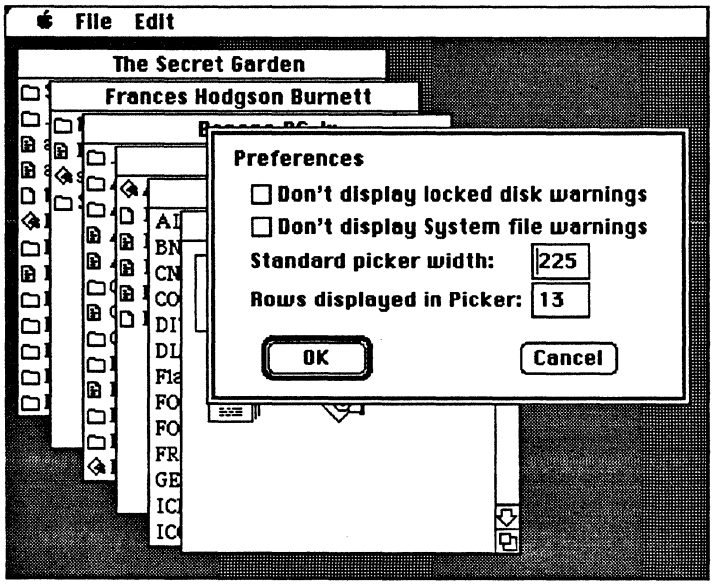
Revert — Changes all resources of the open type back to what they were before you opened the resource type window. Note that this does *not* revert the entire file.

Get Info	Displays resource information and allows you to change it. Figure 2-6 is an example.
Preferences	Allows you to choose whether the "locked disk" and "System file" warnings are displayed, and to set two parameters for the current picker's display. (See Figure 2-7.)
Transfer...	Quits ResEdit, allowing you to transfer control to another application.
Quit	Quits ResEdit. If you have made any changes, ResEdit asks whether you want to save them.

■ **Figure 2-6** An 'ICN#' Get Info window



■ **Figure 2-7** Preferences dialog box



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 as Template, a list of templates is displayed, and you can pick the one you want to use.

◆ *Note:* Using a template with a resource that does not match its definition is improper and may cause errors. Please be careful when you choose a template. (See Chapter 4 for more information.)

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 RESEDIT'S 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. ■

To open an editor for a particular resource in a file, either double-click the resource type name or select it and choose Open from the File menu. 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. All the editors use File and Edit menus similar to those described in Chapter 2, but operate on individual resources or individual elements of a resource, and hence vary in their appearance and function as explained in this chapter.

If you 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; for these resources, 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, else your next mouse-click will turn off the marquee and invert whatever pixel the mouse is on. You can also cut, copy, and paste 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 40x40-bit area from a paint program into, say, an 'ICON', which can only hold a 32x32-bit area), ResEdit beeps and does *not* perform the paste.

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.

Using the general editor

If you hold down the Option key while opening a resource, the general resource editor is invoked. This editor allows you to edit the resource as hexadecimal or ASCII data. The general resource editor can edit resources larger than 255K bytes. If a resource is between 256 and 511K, each click in the up or down scroll arrow scrolls two lines; if between 512 and 767K, 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 can only have values between 0 and 255.)

When you are using the general editor, if you enter hexadecimal text, the editor maintains byte alignment of the incoming data. Thus, if you type `2`, the editor displays `02`. If you then type `A`, the editor displays `2A`.

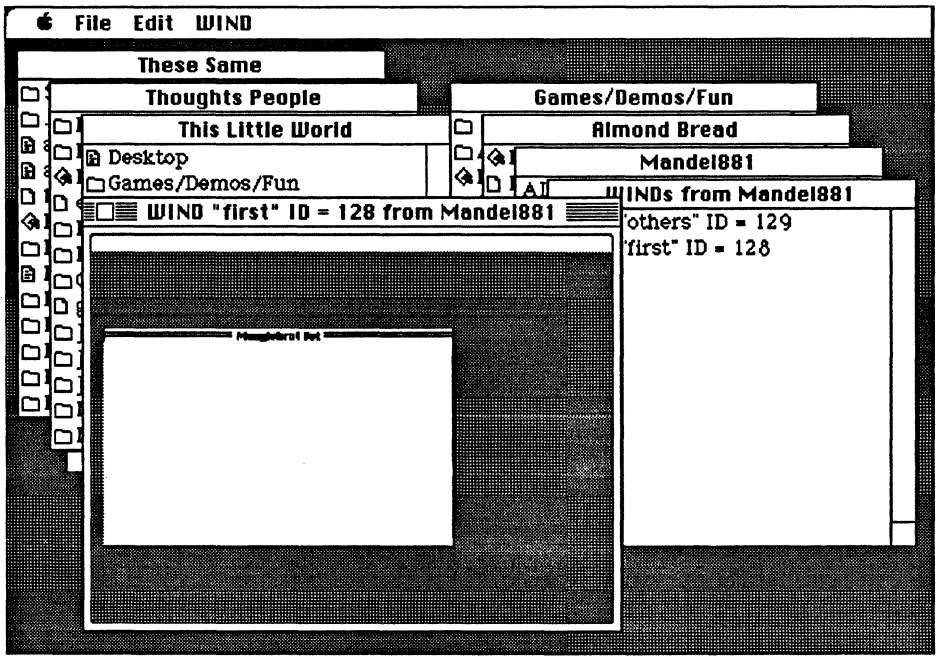
The general editor has a Search menu. It allows you to search for the occurrence of a pattern in the resource being displayed, and allows you to enter the pattern in either hexadecimal or Macintosh character set notation (the latter being loosely described as "ASCII". See Appendix C for a chart of the Macintosh character set). The general editor also allows you to move to a specified offset from the beginning of the resource you're editing.

'WIND' resources

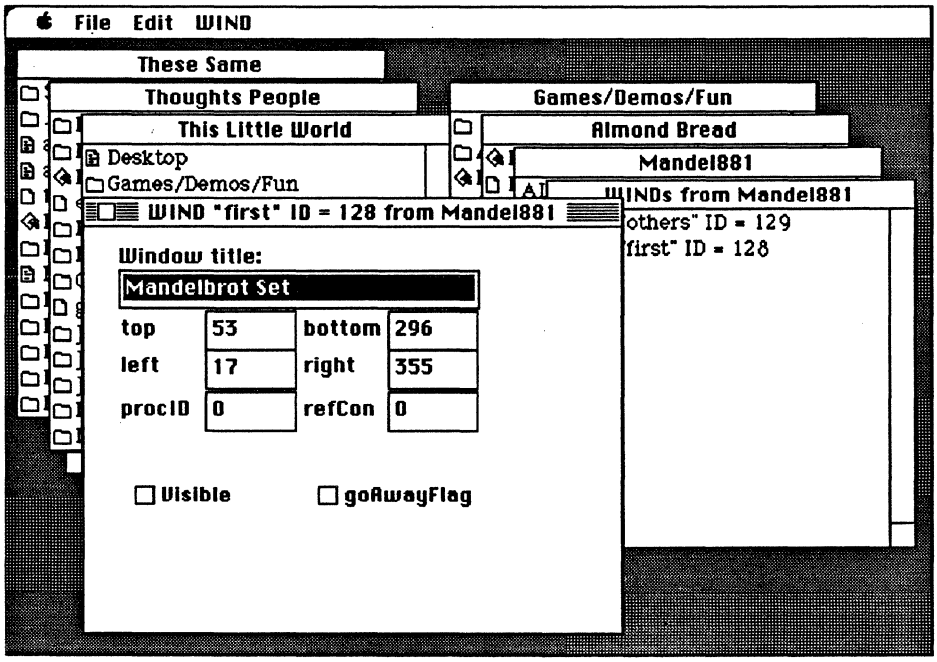
A 'WIND' resource defines a window on the screen. When you open a 'WIND' resource, ResEdit displays a small picture of the screen with the window shown in its usual size and location, to scale, and also presents a special menu, with the title WIND. You can size the window by using its lower-right corner. 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. To change the name of the window, select Display as Text from the WIND menu. (When the window appears on the screen in normal operation, the name may be displayed. If it is displayed, it shows up as a title, in the title bar.)

Figure 3-1 shows a 'WIND' resource open for editing. Notice that there is a 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 3-2 shows the same 'WIND' resource displayed as text.

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



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

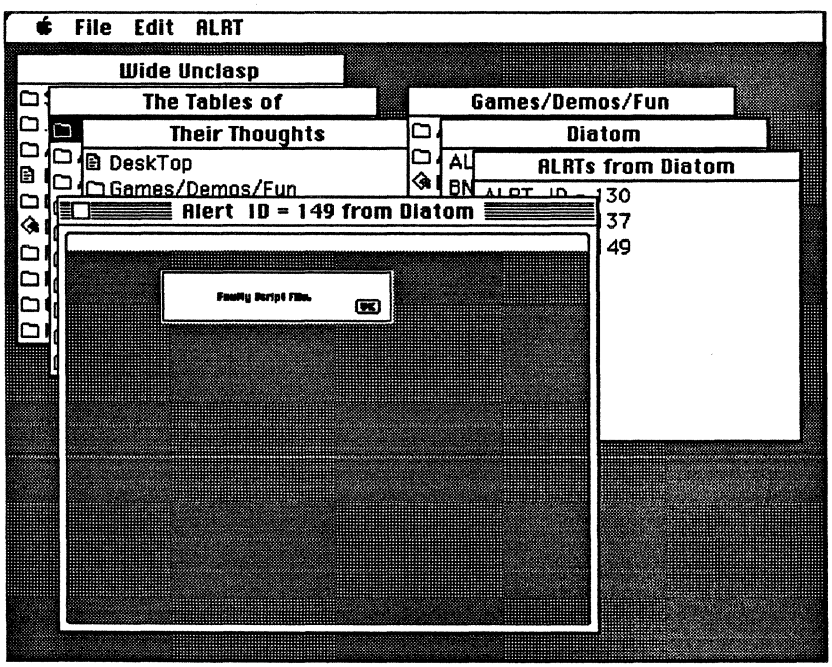


'ALRT' and 'DLOG' resources

'ALRT' and 'DLOG' resources display dialog boxes on the screen. Editing them is much like editing 'WIND' resources, except that if you double-click on the picture of the dialog box after opening the resource, the corresponding 'DITL' resource is automatically opened. (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. In the text view, the resource ID of the associated 'DITL' can be changed.

Figure 3-3 shows an 'ALRT' open for editing. You can see the ALRT menu title in the menu bar. Notice the white area at the top of the window, just under the words *Alert ID = 149 from Diatom*; this space is where the menu bar appears when the alert is displayed on the screen.

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



'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.

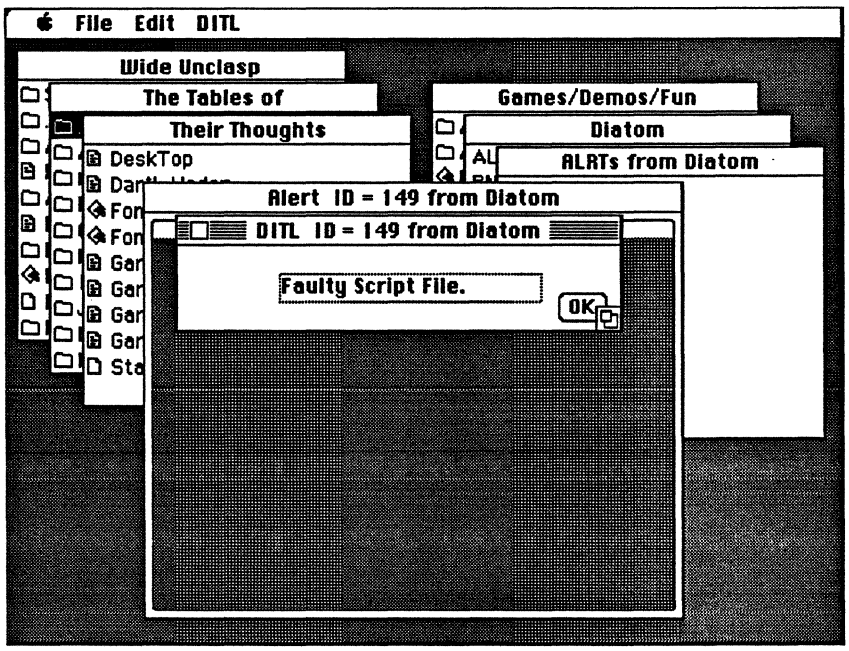
If you open an item within the dialog box, the editor associated with the item is invoked; for an 'ICON', for example, the icon editor is invoked. If you hold down the Option key while opening a 'CNTL', 'ICON', or 'PICT', the general data editor is invoked. If you hold down the Option and Command keys while opening a 'CNTL', 'ICON', or 'PICT', 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 3-4 shows the 'DITL' corresponding to the 'ALRT' from Figure 3-3. The ALRT menu has been replaced by the DITL menu.

■ **Figure 3-4** Editing a 'DITL' resource



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-pixel–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.

Use Owner Window

Changes the 'DITL' back to the size specified in the parent 'DLOG' or 'ALRT'. The algorithm used to find the parent is as follows:

1. Check for a 'DLOG' with the same ID;
2. Check for an 'ALRT' with the same ID;
3. Check for any 'DLOG' that refers to this 'DITL';
4. Check for any 'ALRT' that refers to this 'DITL';
5. Use Full Window.

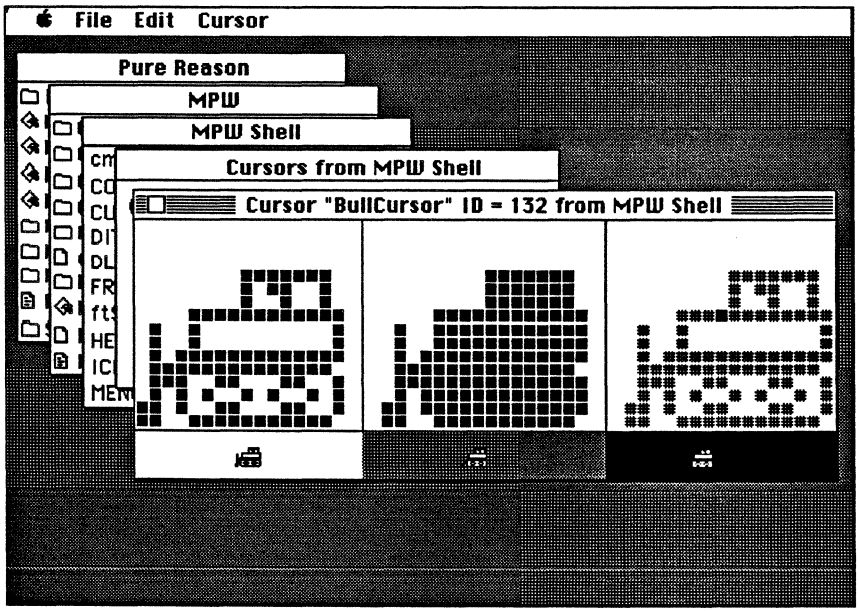
Font and Size menus are also present. These menus are provided to allow you to see how your 'DITL' looks when displayed in various typestyles. The font and size you set by using these menus are not saved, and must be reset each time you edit the 'DITL'.

'CURS' resources

Cursors are pictorial resources of type 'CURS'. Figure 3-5 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 on 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 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 onto the hole. Ordinarily, the mask should just be a filled-in outline of the cursor, so that the cursor can be seen clearly.

■ **Figure 3-5** Editing a 'CURS' resource



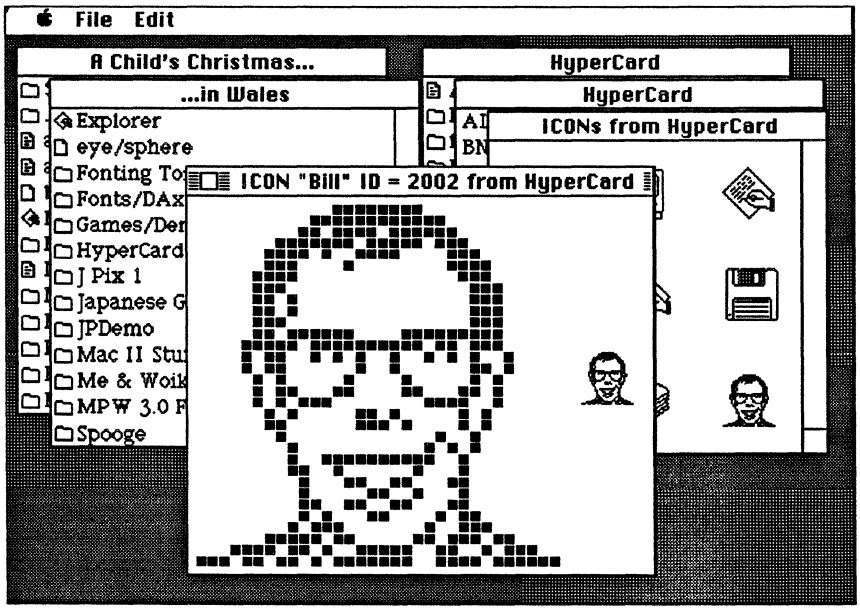
The Cursor menu contains the following commands:

Try Cursor	Lets you try out the cursor by having it become the cursor in use.
Restore Arrow	Restores the standard arrow cursor.
Data -> Mask	Makes a filled-in copy of the cursor in the mask editing area.

'ICON' resources

When icons appear within a program (HyperCard, where it is common to attach icons to buttons, is a good example), they are resources of type 'ICON'. The 'ICON' editor, as shown in Figure 3-6, 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 3-6** Editing an 'ICON' resource



'ICN#' resources

The 'ICN#' resource is one of the most common targets for ResEdit. The icons that you see on the desktop, representing applications and their documents, are all 'ICN#' icons, as are folder icons and even the trashcan. The 'ICN#' resource is edited with a bit editor that permits you to change any of the pixels in the icon, which are in a 32-pixel–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, it doesn't make any sense to talk about moving it. 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.

The 'ICN#' editor displays two panels in the window (Figure 3-7). 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's small icon view.

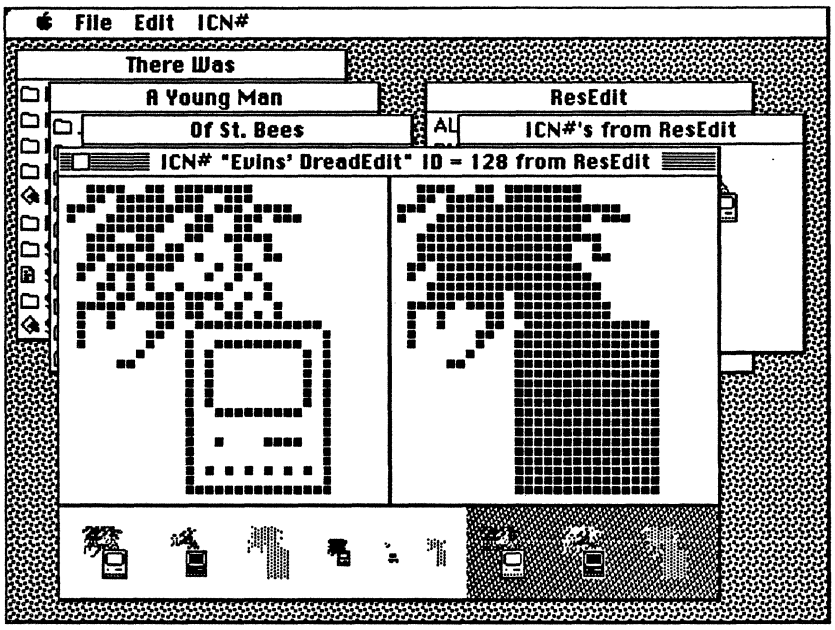
In recent versions of the Finder, 'ICN#' resources are displayed on the screen as follows: First the mask is used to blank an area of the screen. Then an OR operation is performed, using the icon as data, in the same screen area. (When a highlighted icon is displayed, the foreground and background colors 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 pre-6.0 Finders. 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.

■ **Figure 3-7** Editing an 'ICN#' resource

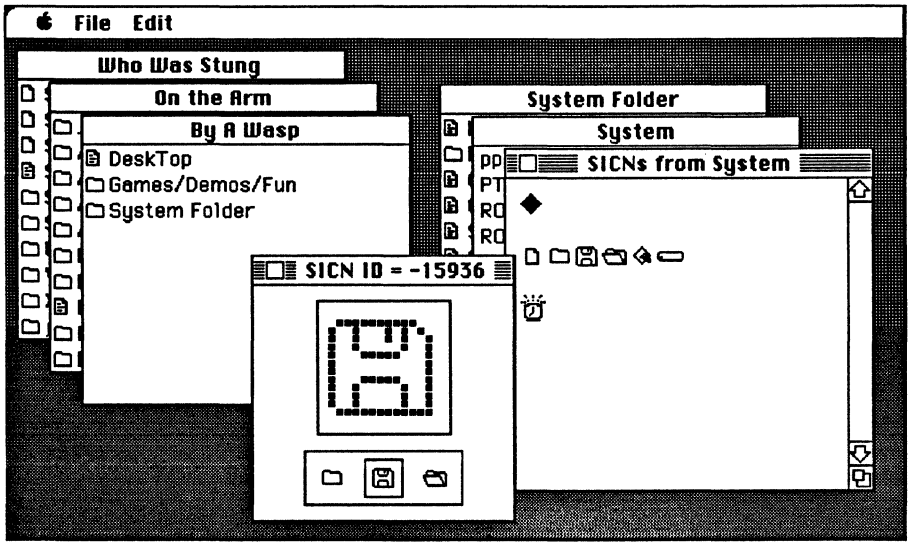


'SICN' resources

Small icon ('SICN') resources are edited with a bit editor, just as other pictorial resources are. Unlike 'ICON' or 'ICN#' resources, 'SICN' resources can, and usually do, occur in groups. A typical display is shown in Figure 3-8. The upper panel is enlarged, and shows the icon currently being edited. The lower panel shows three 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 righthand or lefthand 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 3-8** 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 C.

Because the Macintosh has no text mode, 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.

The Macintosh can modify elements of a font, to slant them for an approximation of italics, embolden them, and so on. 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. Frequently, 'FONT' resources come in families, so that it is possible to display text on the screen (and print it on dot-matrix printers) in several styles, 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 release, 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 is a very large number of bitmap fonts (many 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.

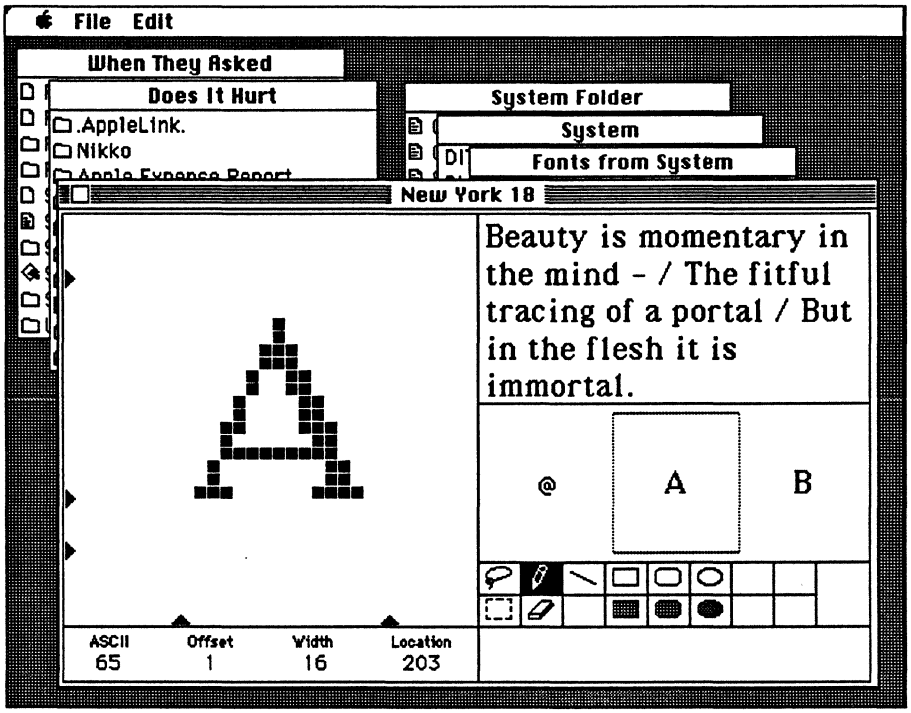
There is also another, newer kind of font resource, type 'NFNT'. Like 'FONT' resources, 'NFNT' resources are also owned by 'FOND' resources. ID numbering of 'NFNT' fonts is, however, not keyed to the ID number of the parent 'FOND'. Arbitrary numbering of 'NFNT' resources helps avoid font ID number collisions, and facilitates resolution of conflicts when they do occur. 'NFNT' fonts can contain and display more than one 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 can copy and move them, as can version 3.8 and later versions of the Font/DA Mover program. 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 3-9.

■ **Figure 3-9** 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 (that is, the character width). Two of the three triangles at the left side of the panel control the ascent and descent. 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 ASCII is minimal. The Macintosh character set is shown in Appendix C.

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 lassoo is also available.

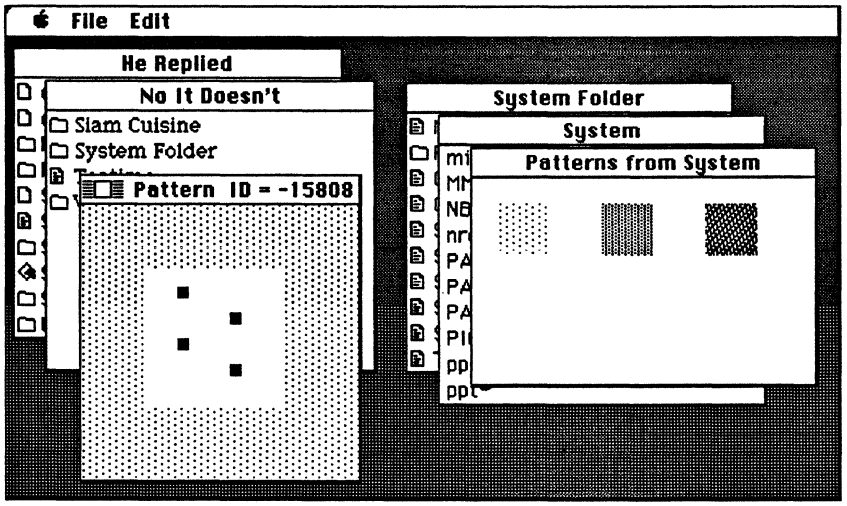
Any changes you make in the character editing panel are reflected in the text panel and the character selection panel, except on monitors displaying more than 2 colors or gray levels.

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 3-10. It displays a single panel, with the pattern shown around a central editing area. This area shows the pattern, enlarged. The outer area shows the pattern at full scale, displaying changes as you edit. 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.

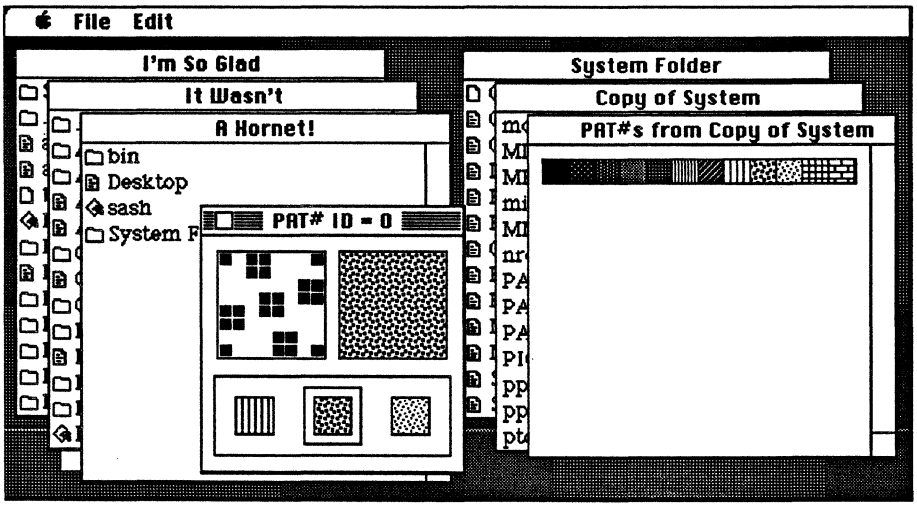
■ **Figure 3-10** Editing a 'PAT ' resource



'PAT#' resources

The 'PAT#' resource (pattern list) editor is a bit editor, much like the 'SICN' editor, and is shown in Figure 3-11. 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.

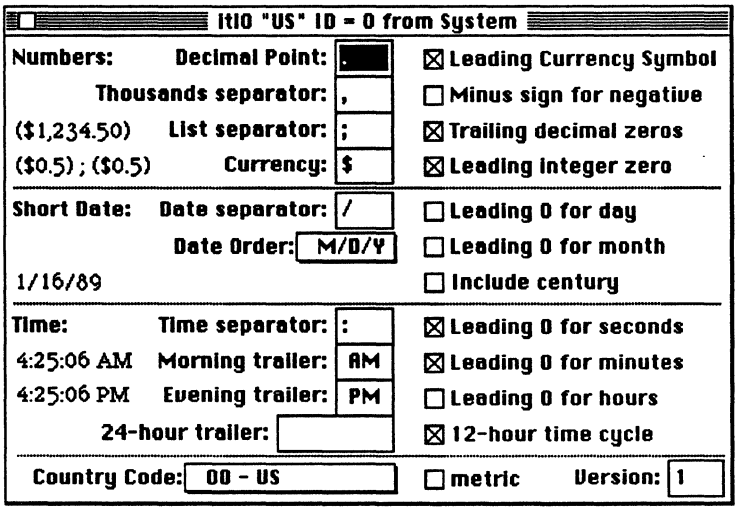
■ **Figure 3-11** Editing a 'PAT#' resource



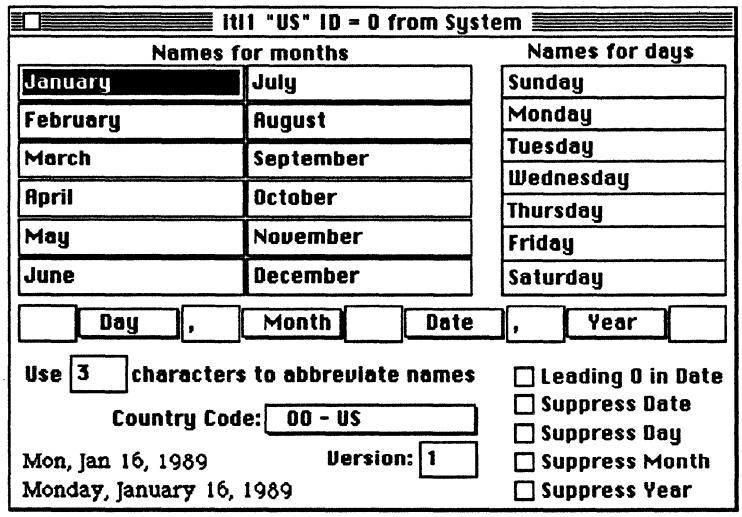
'INTL', 'itl0', and 'itl1' resources

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 3-12 and 3-13 show the windows for 'itl0' and 'itl1'.

■ **Figure 3-12** Editing an 'itl0' resource



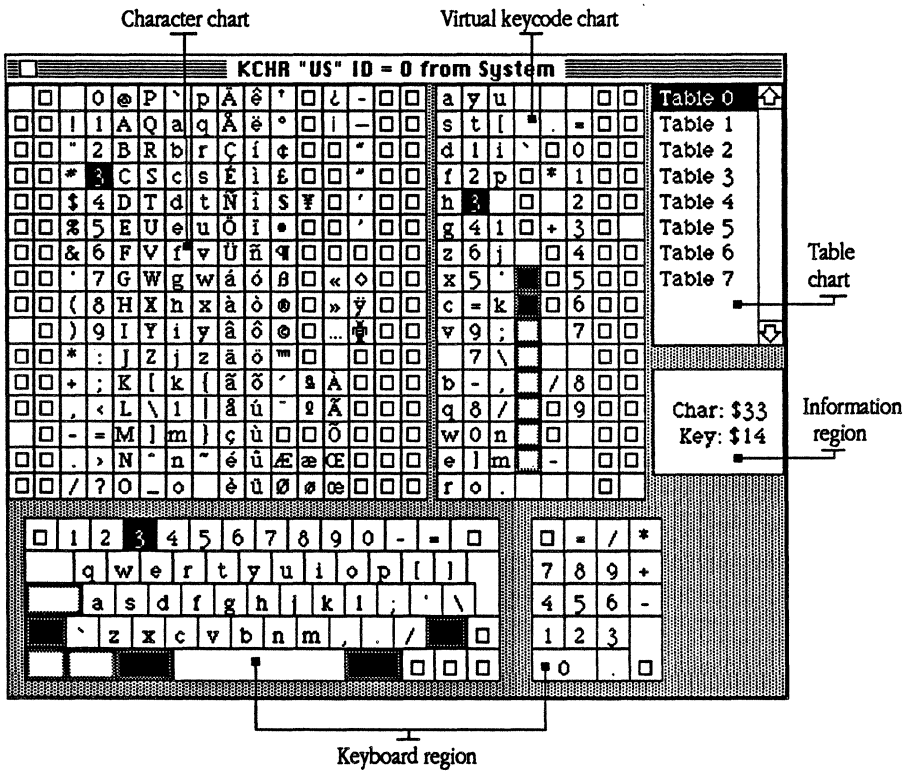
■ **Figure 3-13** Editing an 'itl1' resource



'KCHR' resources

The 'KCHR' resource controls keyboard mapping. The 'KCHR' editor can be used with any Macintosh that runs system software release 5.0 or later. The main 'KCHR' editing screen is shown in Figure 3-14, with Command-Option-3 pressed; the "dead" key editor is shown in Figure 3-15. There is an in-depth discussion of the 'KCHR' resource itself in Appendix A, and a short section of 'KCHR' questions and answers in Chapter 6.

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



The main 'KCHR' editor

The display for the main 'KCHR' editor (Figure 3-14) is divided into five parts, 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 notice that although there are 256 possible combinations of modifier keys, most versions of the 'KCHR' resource use only 8 tables, and very few would 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 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 has two sets of modifier keys, and you can use the "Uncouple modifier keys" command 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 connected to your Macintosh, you cannot choose "Uncouple modifier keys."

Note that the modifier keys shown on 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 also have a gray border there.)

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.

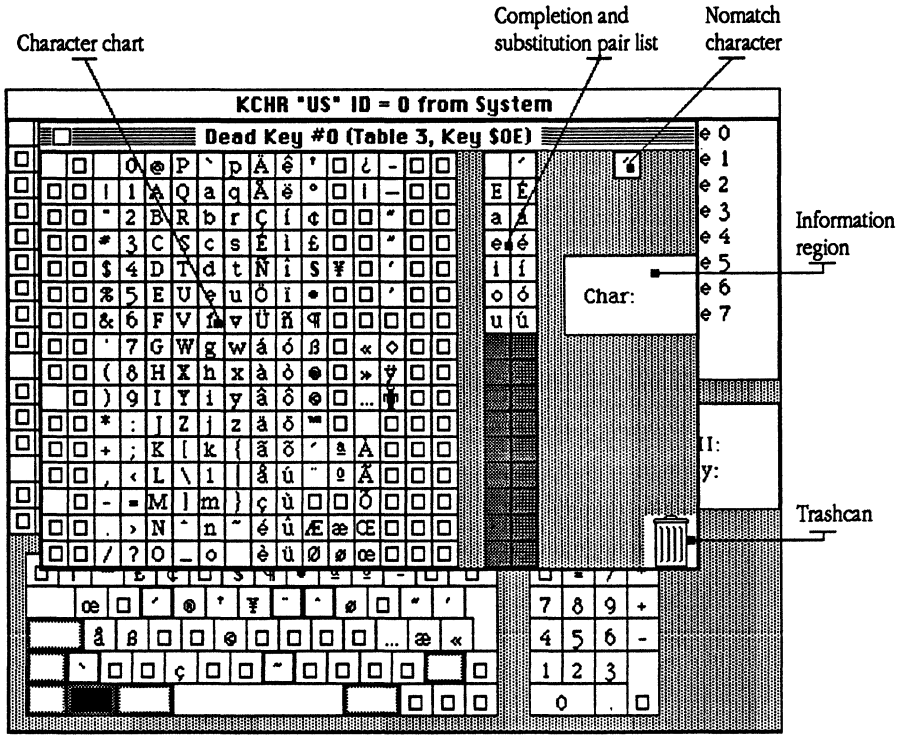
The information region

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 3-15.

■ **Figure 3-15** 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. The left column shows all completion characters; the right column 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 trashcan

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 in one of the other parts of the editor.

The menus

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

The 'KCHR' menu

This 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

When you have an ADB extended keyboard connected to your computer this command is enabled. 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 Uncouple modifier keys, you must also use View as 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 to see if there are tables that are not used by any modifer key combination, and removes them.

Remove duplicate tables

Looks to see if there are some tables that are completely 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.

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. ■

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 in ResEdit that is named LAYO.) The template specifies the format of the resource and also specifies what labels should be put beside the editText items in the dialog box that's 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 a bit 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 early 1989, ResEdit contains 'TMPL' resources for these resource types:

'actb'	'CTY#'	'finf'	'itlk'	'PICT'	'STR '
'acur'	'dctb'	'FOND'	'LAYO'	'pltt'	'STR#'
'ALRT'	'DITL'	'FONT'	'MACS'	'ppat'	'TEXT'
'APPL'	'DLOG'	'FREF'	'MBAR'	'PRC3'	'TMPL'
'BNDL'	'DRVR'	'FRSV'	'mctb'	'PSAP'	'vers'
'cctb'	'FBTN'	'FWID'	'MENU'	'ROv#'	'wctb'
'clut'	'FCMT'	'insc'	'minf'	'scrn'	'WIND'
'cmnu'	'fctb'	'itlb'	'nrct'	'SIGN'	
'CNTL'	'FDIR'	'itlc'	'PAPA'	'SIZE'	

Editing

When you are editing a template, the Tab key moves you from field to field within the template. 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 New from the File menu.

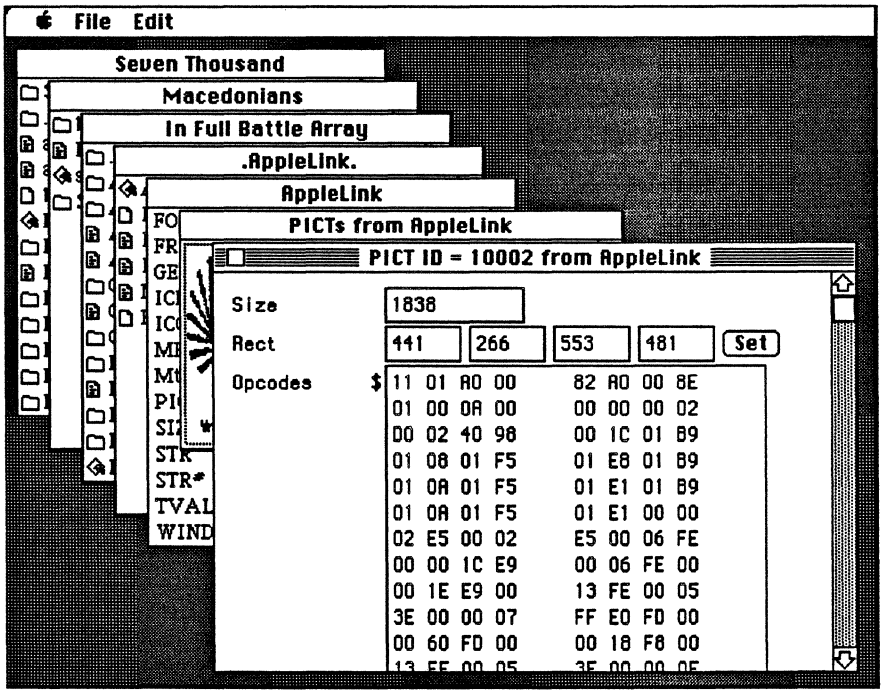
Some templates control windows or other rectangles. Frequently such a template will have a Set button that lets you draw a rectangle on the screen. The pixel numbers for the rectangle are automatically copied to the appropriate fields in the template. There is a Set button in the 'LAYO' template, which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

Values can be entered into numeric fields in either decimal or hexadecimal notation. Hexadecimal numbers are preceded by 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 'WIND' 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.

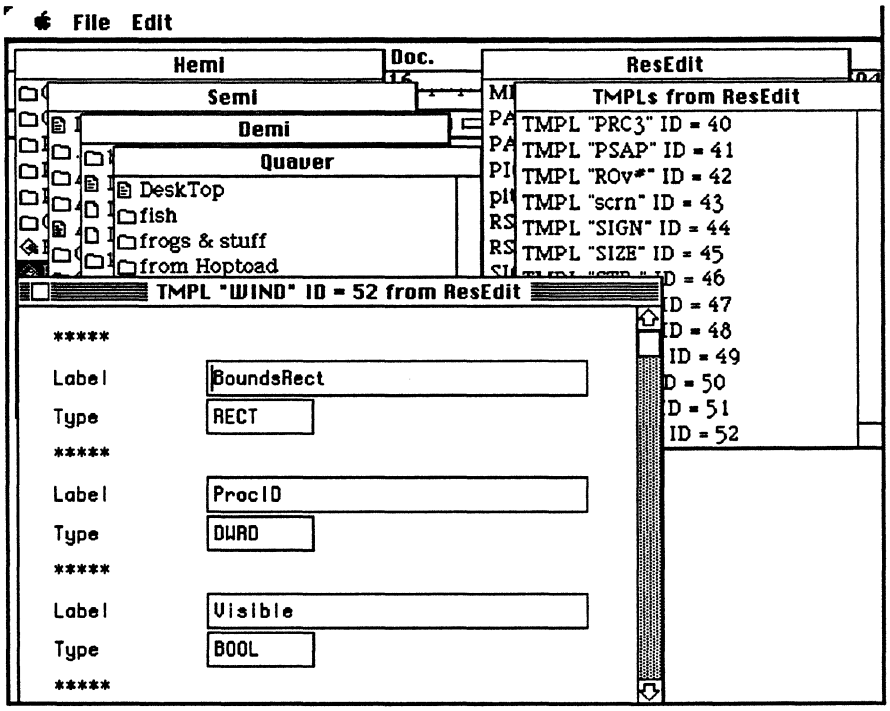
Chapter 5 **Creating ResEdit Templates**

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 'TMPL' resource inside ResEdit with name WIND is shown in Figure 5-1. It is shown here as a ready example of what 'TMPL' innards look like on the screen. The contents of this window continue beyond what is visible in the figure, as you can tell by the scroll bar at its right edge.

Ordinarily, of course, 'WIND' resources are edited with a custom editor. You can edit them with the template shown in Figure 5-1, though, if you care to.

■ **Figure 5-1** 'WIND' template data



The window template consists of the following elements:

- A RECT (four words) specifying the boundary of the window.
- A word that is the procID for the window. (DWRD tells ResEdit to display the word in decimal as opposed to hex.)
- A Boolean indicating whether the window is visible. (BOOL is two bytes in the resource but is displayed as a radio button in the dialog window used for editing.)
- Another Boolean indicating whether the window has a close box.

- A long word that is the reference value (`refCon`) for the window. (DLNG indicates that it should be displayed in the editor as a decimal number.)
- A Pascal string (PSTR), the title of the window.

You can look through the other templates and compare them with the structure of those resources to get a feel for how you might define your own resource template. (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, word, long word.

HBYT, HWRD, HLNG
: Hex byte, word, long word.

AWRD, ALNG
: Word , long align.

FBYT, FWRD, FLNG
: Byte, word, 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 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 (where *nnn* < $900); displays *nnn* bytes in hex format.

◆ *Note:* Scrolling can become extremely slow if there are a lot of BBIT or BOOL items in a template.

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.

Remember that the list end is signalled by a NULL byte. There is a bug in ResEdit that causes it to think the end of the list has been reached if you have any field data that *begins* with $00. Please be careful!

Your template does not have to be inside ResEdit; it can be in any open file. 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 New command to create the 'TMPL' type if it doesn't already exist in the file.
3. Choose New from the File menu.
4. Select the list separator (*****) by clicking it with the mouse.
5. Choose New from the File 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 File 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. Herein are presented 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. Also, if you edit a file that is already open, for example the current System file, and you make any changes, you *must* save those changes!
- If you choose Get Info for ResEdit, you will find that Application Memory Size is set to 512K. If you are editing very large resources, even 512K is not sufficient. On the other hand, ResEdit can be run in as little as 384K if necessary, particularly if you are only dealing with small resources.
- 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 immediately previous tip.)
  2. Use Copy to put the ID number of the new 'PICT' in the scrap.
  3. Go to the 'DITL' that belongs to the 'DLOG' you are adding the picture to.
  4. Choose New Item.
  5. Click the PICT button.
  6. Paste the ID number from the scrap.
  7. Close the Dialog Item Editor.
  8. Choose Use RSRC Rect from the menu.
  9. 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 close the 'ICN#' or 'ICON' editor and picker. (You must close the picker in order for this procedure 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 perform the paste operation.
- There are keyboard equivalents for many operations you would ordinarily perform with the mouse. Try selecting a file 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. The arrow keys also work—for example, in a file list, you can go down the list with the down-arrow key, and you can even select an individual resource by typing its ID number.
- 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:
  - At the individual resource level: Option–doubleclick for Open General.
  - At the individual resource level: Option–Command–double-click for Open As Template.
  - At the individual resource level: Option–Command–Shift–double-click (or Shift–Open As Template) displays the template-type dialog box without the list of templates. (You can type in 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. This feature does not work at the individual item editor level, even in the general or template editors. 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.)
  - Option-open a resource type to use the general picker.
  - Shift-New to create a new resource type gives you the "new type" dialog box without the list of resources. You must, of course, type in the resource type you want, rather than being able to select from the list. Again, 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). 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.

- Depending on where you use it, Revert does different things. In an editor, it reverts only the currently open resource. In a file window, it reverts the entire file. In a resource picker, it reverts all resources of the current type.
- 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.
- 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 put them 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 to, mark it as invisible, and reposition it exactly in your code. Have your code mark it visible right after displaying it. (This avoids embarrassment.)
- 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.

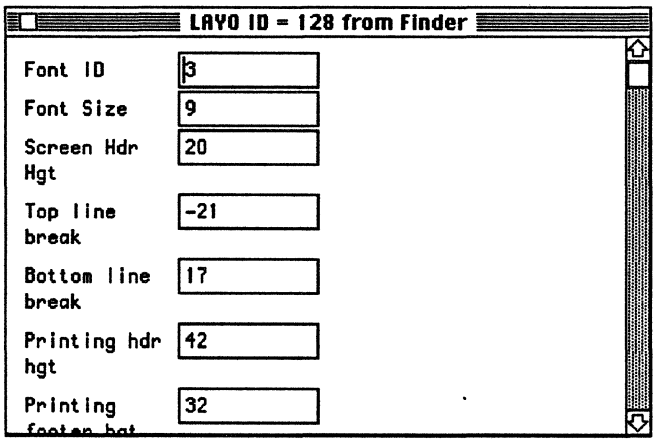
The 'LAYO' resource

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 you are under MultiFinder and you try to open the currently active Finder, you get an error message that tells you 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. 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 (for Times at 10 points, as the 9-point version of Times is very small).

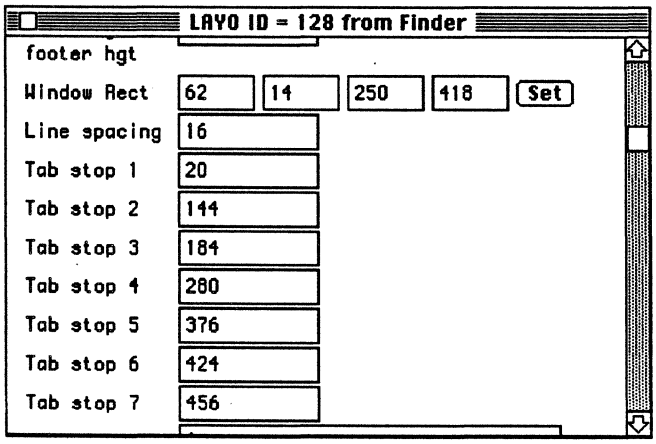
■ **Figure 6-1** 'LAYO' template, view 1



The line of numbers labeled Window Rect in Figure 6-2 allows you to specify the default folder (and disk) window size and location. 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 you are running under MultiFinder, editing 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, the obvious result will obtain: You will find yourself summarily ejected from ResEdit into whatever you have clicked on. The cure is equally obvious: Move a ResEdit window to the area where you want to start drawing your rectangle before you click on the Set button, or use the number fields instead of the Set button.

You can also explicitly set the location of seven tab stops.

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



A bit further down the template are the numbers that control the placement of the icons themselves, as shown in Figure 6-3. Some people dislike having icons with long names overlap and obscure 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 6-3** 'LAYO' template, view 3

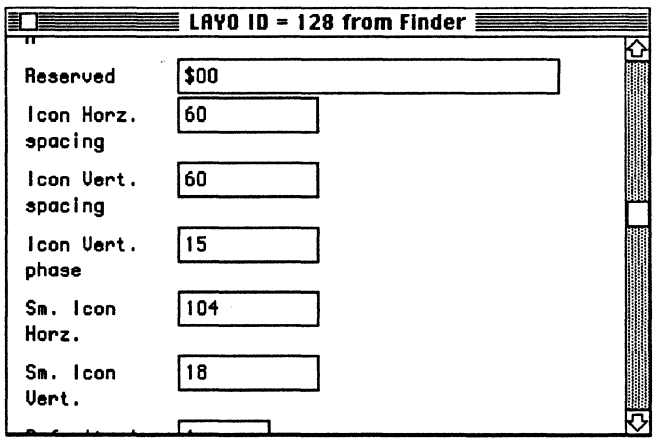


Figure 6-4 shows some unused bits, and three commands, the first of which ("Use zoom Rects") is on by default. If you set it to False, the Finder will not use the zoom boxes that are present in many windows.

"Skip trash warnings" prevents the system from asking you 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 will eschew this option.

The Watch Thresh setting (not visible in any of the pictures) 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 6-4** 'LAYO' template, view 4

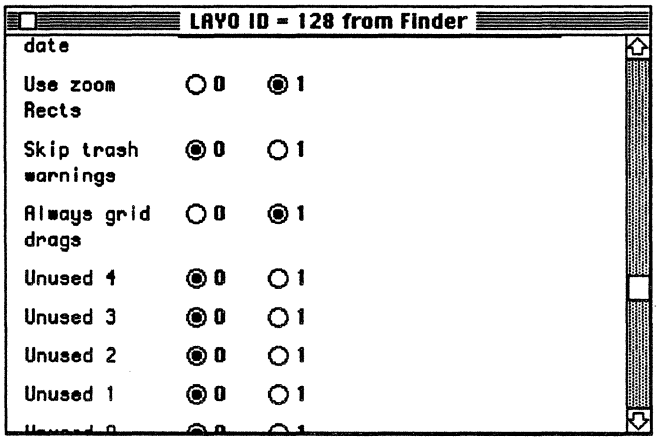


Figure 6-5 shows a few more unused bits and the end of the template.

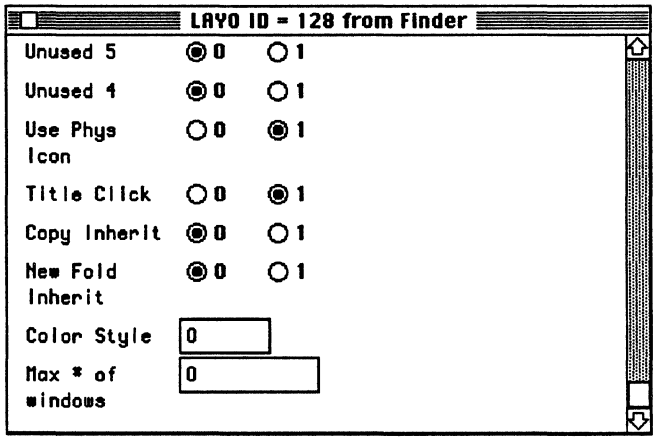
Use Physical Icon is handy if you have a Macintosh II or Macintosh SE with two floppy disk drives. If this option is on, the icon you get when you insert a floppy disk into your machine indicates which drive the floppy disk is in. The disk location is certainly easy enough to recall just after you put the disk in, but you may forget it later. Not a major issue, but certainly a pleasant convenience.

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 AppleTalk® 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.

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



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 drag a character (with the mouse) 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 that was 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 which was added to the end of the list (while still holding down the modifier keys).
4. Choose OK in the alert 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 the editor is closed, 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 the key is being held down. 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.

■ **In the dead-key editor, how do I create a new completion/substitution pair?**

When the dead-key editor is active, you may 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.

■ **In the dead-key editor, how do I delete a completion/substitution pair?**

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 **A Development Scenario**

THERE ARE SEVERAL USES OF RESEDIT that become very familiar to developers, with practice. This chapter purports to show you one of them, to make the path a little less brambly for you, so you don't need to sharpen your machete quite so often and so your arms don't get quite so tired. ■

Putting an icon on your application

Let us suppose for a moment that you are Josephine Developer, and you want your shiny new application and its documents to have their own icons, so they will have a distinctive look on the screen. In order to add icons, you must provide the icons themselves, the information that the Finder needs in order to know that your application has its own icons, and the information that tells the Finder which icon to match with each file type.

The 'BNDL' resource describes how file types match up with icons.

The first field in the 'BNDL' editing template is for the "signature" of your application. This field is labeled OwnerName. The signature consists of four characters that are used as the creator type for your application and its documents. (Figure 2-2 shows the information display window for a file. The creator type is at the right side of the window.) Your signature may be any combination of four characters, so long as at least one of them is an uppercase letter. Apple reserves all signatures without any uppercase letters. You may include space, Option-space, and punctuation symbols, but they don't count as upper case letters.

The second field in the 'BNDL' template is labeled OwnerID, and must be zero.

The 'BNDL' can have any ID, but 128 is a typical value. The 'BNDL' lists the 'FREF' and 'ICN#' resources that will be used. It associates a local ID with an actual resource ID for each resource. For example, in Figure 7-1, the 'BNDL' refers to two 'FREF' resources and two 'ICN#' resources. The local IDs of the 'FREF' resources will be used by the Finder. The local IDs of the 'ICN#' resources will be used by the 'FREF' resources to tell which icons to show for which file types. (The term "local ID" refers to an entry in a resource table inside a resource, in this case the 'BNDL' resource.)

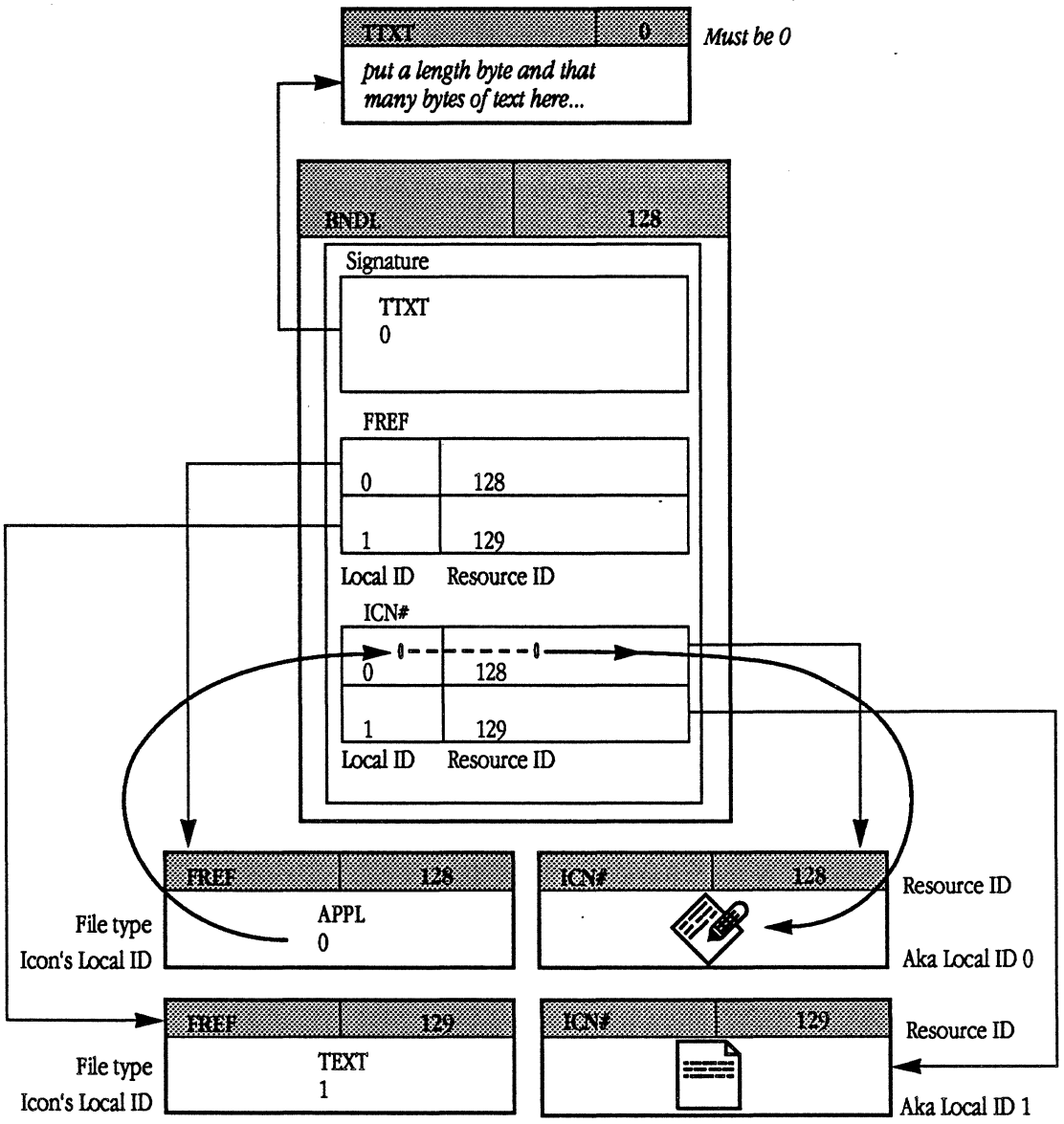
One 'FREF' resource exists for each file type to which you want to attach an icon. For example, you could have text documents, read-only text documents, and, of course, the application itself. Every 'FREF' has a different ID and a different file type value in it. The icon local ID value in each 'FREF' does not have to be unique. It must, however, match one of the 'ICN#' local ID values listed in the 'BNDL' resource.

You need one 'ICN#' resource for each different icon you want to show. While it is typical that one 'ICN#' resource exists for each 'FREF' resource, you don't have to do it that way. For example, you may not care to have a separate icon for read-only text documents. You might just want them to have the regular text document icon.

You also need a signature resource. Its type is the same four characters as the signature in the 'BNDL'. The signature resource consists of a Pascal string that lists version and copyright information. The ID of this signature resource must be zero.

In all, four resources are necessary for the Finder to recognize and show your application's icon. Figure 7-1 shows a slightly more involved example which has an icon for itself, and an icon for its documents. The four characters 'TTXT' are the creator type for this particular application. In Figure 7-1, the 'BNDL' resource lists all of the other resources involved ('FREF', signature, and 'ICN#'). On the right are the five additional resources listed in the 'BNDL'.

■ **Figure 7-1** Six resources and their relationships



Once you have created these resources inside your application, set the application's bundle bit. The set bundle bit tells the Finder that it should look for the 'BNDL'.

The icon has to be in the Desktop file in order for the Finder to display it. To be sure the icon gets put into the Desktop file, copy your application to a floppy disk, and rebuild the Desktop file on the floppy disk. To do that, hold down the Option and Command keys and insert the disk into your Macintosh. You will see a dialog box that asks whether you really want the Desktop rebuilt. Answer OK. If you have done everything correctly, you should see your application's new icon. If you change your icon, rebuild the Desktop file in order to see the new one.

Copy your application onto a floppy and, if necessary, rebuild the Desktop file on the floppy disk. (To do that, hold down the option and command keys and insert the disk into your Macintosh. You will see a dialog box that asks whether you really want the Desktop rebuilt. Answer Yes.) Your application should now have its own icon.

As a general rule, your resource IDs should always be in the range 128 through 32767. Resource numbering is discussed in more detail in *Inside Macintosh,* Volume I, Chapter 5. Further discussion about adding icons to your applications and documents is offered in *Inside Macintosh,* Volume III, Chapter 1, and in Macintosh Technical Note #48.

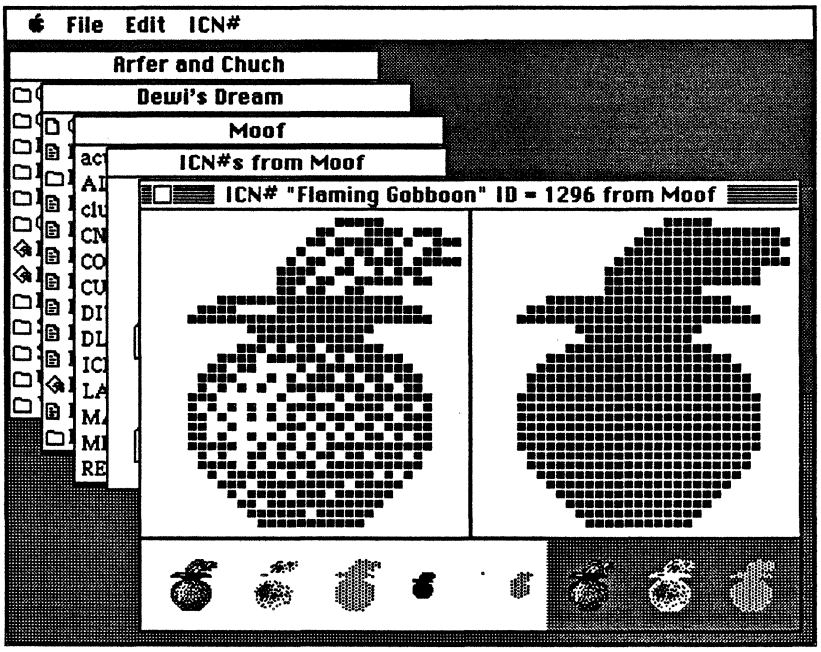
Let's try that once again, from the top

Here are the steps for a specific example, in which we attach an icon to an application.

- Create an icon that has the look you want, and get it into your application.

1. Find some 'ICN#' that looks vaguely similar to what you want, or has minimal stuff in it. Else, start from scratch by choosing the New command.
2. If you are starting with an existing icon, copy it, then paste it into your application. If you are using New, you can create the icon inside your application directly.
3. Edit the icon until you have what you want. (See the 'ICN#' editing section in Chapter 3 for details of this process.) Results are shown in Figure 7-2. While this is probably not the distinctive look you actually want for your application, it *is* different from most icons you find lying about in the street.

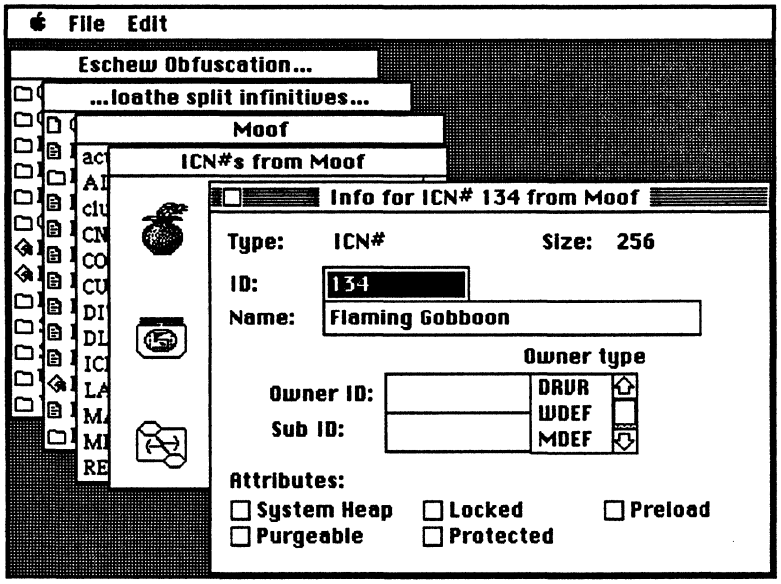
■ **Figure 7-2** Edited 'ICN#', ready to go



■ Give the icon an appropriate ID number.

1. Do a Get Info on the icon and choose an ID number that makes sense to you—and, if you like, a name. (ID numbers here must be at least 128.) Figure 7-3 shows the info for the 'ICN#' shown in Figure 7-1.

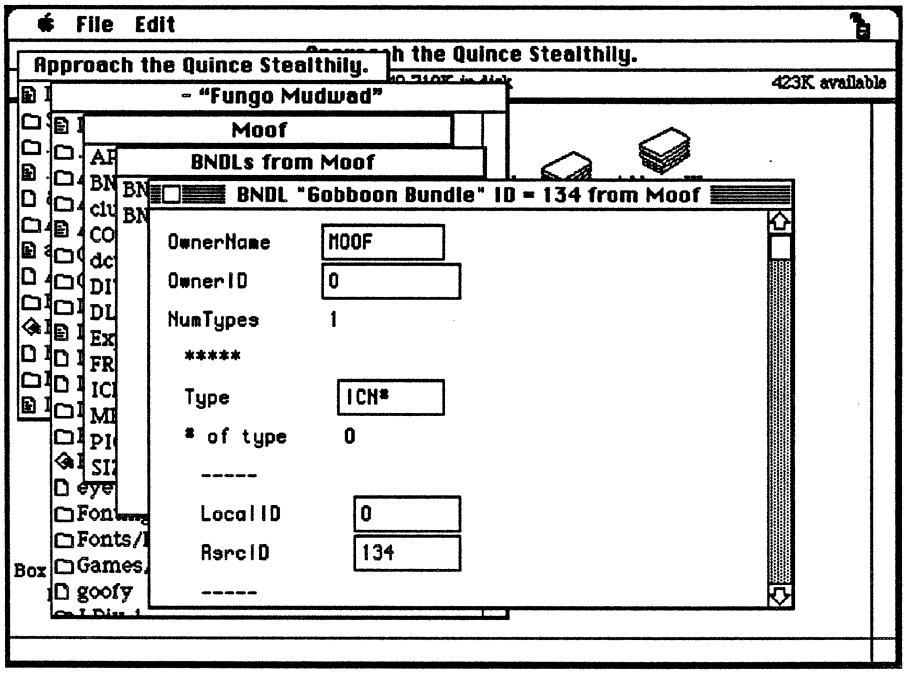
■ **Figure 7-3** 'ICN#' info box



■ Create a 'BNDL' resource and a 'FREF' resource to match.

1. Copy a 'BNDL' resource and a 'FREF' resource and paste them into your application, or create your own from scratch. You will, of course, have to do them one at a time, unless you have a file that has one each 'BNDL' and one each 'FREF'.
2. Configure the 'BNDL' to match the 'ICN#'. Figure 7-4 shows the top of the template of an appropriate 'BNDL'. (The bottom of the template, not visible in the figure, lists a 'FREF', also with resource ID number 134.)

■ **Figure 7-4** 'BNDL' template view



3. Edit the 'FREF' similarly. It is actually very simple, and merely needs to note that your File Type is APPL, and your Icon localID is 0. If your application has documents associated with it, and you want to have icons for those too, you will need more 'FREF' and 'ICN#' resources to go with them. Look at the 'FREF' resources in a reasonably complex application for an example.
4. Create the signature resource. Use the New command at the file level to make a new resource, ID 0, the type name of which is the OwnerName from your 'BNDL'. In this case, that's 'MOOF'.
5. You probably want to put a Pascal string into the empty signature resource. Include the version and copyright information and anything else you deem appropriate. (A Pascal string consists of a length byte followed by that many bytes of text.)

- Make sure it's all turned on.

1. Use ResEdit to do a Get Info on your application, and set the Bundle Bit.
2. Use the Finder to do a Get Info on your application.

If you don't see your icon, there is another step you can take to get the icon into the Desktop file so you can actually see it.

Copy your application onto a floppy and, if necessary, rebuild the Desktop file on the floppy disk. (To do that, hold down the Option and Command keys and insert the disk into your Macintosh. You will see a dialog box that asks whether you really want the Desktop rebuilt. Answer OK.) Your application should now have its own icon.

Chapter 8 **Extending ResEdit**

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, substituting your own program for parts of its code. This chapter describes the process and discusses necessary and optional functions and procedures. ■

Pickers and editors

When ResEdit uses your program, it looks for two general capabilities: a picker and an editor. Pickers and editors are separate from the main code of ResEdit 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's a suitable editor, it should launch the 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. For instance, a dialog editor might open an icon picker so that you could 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. Sub-editors (described under "GiveSubEBirth," 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' sub-editor in an 'RSSC' resource with the name $DITL.

Samples

A sample resource editor, picker, and LDEF (list definition procedure) are included with ResEdit. The samples are provided in both C and Pascal and use the MPW 3.0 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 ResXXXXEd. 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 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 call.

ResEdit determines which editor should receive which events, so you need to do very little event decoding in 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 ICONPick. The sample picker is the actual 'ICON' picker from ResEdit. The 'ICON' LDEF (in the file ICONLDEF) 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 GNRLLDEF. 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 general LDEF from ResEdit. This LDEF is used to display a files resource list. By looking at this LDEF and the one included with the 'ICON' picker, you can see two different ways of using custom LDEFs.

Building the examples

You can build the examples by using the build scripts provided in the folder appropriate for 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 are experimenting with changing any of these files, you may want to build into a copy of ResEdit. If anything goes wrong, you can get a fresh copy of ResEdit for 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. Your unit 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 ResEd68k.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 in `EditBirth`, `DoEvent`, and `DoMenu`, call

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

if you are writing in Pascal, or

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

if you are writing in C.

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. 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: opening menus, dragging windows, closing windows, switching between windows, and converting double-clicks 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 under "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. The Set Preferences, Quit, and Transfer commands in the File menu are also handled by the main program. The Quit and Transfer commands display the Save Changes dialog box and may pass a Close command to all editors and pickers. (All other commands are passed directly to the `DoMenu` procedure.) If your editor needs to do some cleaning up before the Quit command completes (such as, for example, restoring the system color palette), 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 (menu, item, object);
```

If your picker has loaded all of the resources , the `DoMenu` procedure should release them when a Close command is received (to avoid heap fragmentation). The 'CURS' picker, for example, must load all of the 'CURS' resources so that they can be drawn.

Using custom LDEFs

Usually, you will want to write your own picker simply to display the resource list in a more meaningful way (such as, in the 'ICON' picker, drawing the icons themselves 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 `WindList` in your `PickBirth` procedure, pass the resource ID of your picker as the `drawProc` 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 List Manager chapter in *Inside Macintosh*, Volume IV for details of how the `drawProc` mechanism works.

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 used to indicate that a window's data (for example, a picker's list) needs to be recalculated at the next opportunity. The rest of the handle can be of any format. 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 very 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 under "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: Str64;
    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;            { Any type is OK here }
PickHandle = ^PickPtr;
PickRec = RECORD
    father: ParentHandle;      { Back ptr to dad }
    fName: STR64;
    wind: WindowPtr;           { Picker window }
    rebuild: BOOLEAN;          { Flag set by son to indicate that }
                               { world has changed so father}
                               { should rebuild list }
    pickID: INTEGER;           { Resource ID of this picker }
    rType: ResType;            { Resource type for this 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 procedure }
    scroll: ControlHandle;     { Scroll bar }
    END;
```

Other routines

You are likely to want more than just the required routines in your code. Here are others you can use.

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 as Template was chosen or an editor is not found, the 'GNRL' (template) editor is started. If Open General is chosen or neither an editor nor a template is found, the general (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 a superset of 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 as Template or Open General 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 general 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' sub-editor resource is $DITL. This 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 the same as any other windows.

Information-passing routines

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

`CallInfoUpdate` passes an info 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 info 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 sub-editors 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 by `GetWindowTitle`. You don't need to call this function if you are using the `WindSetup`, `CWindSetup`, or `EditorWindSetup` procedure.

◆ *Note:* You should call `AlreadyOpen`, to avoid opening the same resource twice. `AlreadyOpen` depends on your setting up the `windowTitle` and `windowName` correctly. For pickers, the `windowTitle` must uniquely identify the window. For editors, the name 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>.`

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

This function is identical to `WindSetup` except that it creates a color window.

```
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 the icon picker uses) 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' or Cursor) 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, which 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 List Manager chapter 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);
```

`WindOrigin` moves the window pointed to by `w` to a position below and to the right of the front window. This routine will guarantee that, if possible, the entire window will be visible. If you are using the `WindSetup`, `CWindSetup`, or `EditorWindSetup` procedure, you don't need to call this procedure.

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

`WindSetup` should be called by pickers from the `PickBirth` routine to set up their window(s). It creates and automatically positions a new window with the given width and height, and displays a title formed by the concatenation of `myType` and `name`. A pointer to the window is returned. Calling this routine is equivalent to calling `GetWindowTitle`, `AlreadyOpen`, `WindAlloc`, `NewWindow`, `WindOrigin`, and `ShowWindow`. The `myType` parameter should contain the resource type, and the `name` parameter should contain the name from the father's parent record. For example, the 'CURS' picker makes the following calls at the beginning of its `PickBirth` procedure:

```
GetStr(fromStr, miscStrings, from); {get the 'from' string}
myTitle := 'CURS';
ConcatStr(myTitle, from);
myWind := WindSetup(GetMyWidth, GetMyHeight, myTitle, dad^^.name);
```

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

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

EditorWindSetup should be called by editors from the EditBirth procedure to set up their windows. It is very similar to WindSetup except that it automates a few more details of creating an editor's window. If the color parameter is TRUE, a color window is returned. 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.

- *Note:* NIL is returned if the window can't be allocated for some reason or the window is already allocated (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, with a couple of additions. If an error is detected, an alert is displayed and FALSE is returned; TRUE is returned otherwise.

```
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 procedure would be 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`, described earlier, 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 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), 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. Other than this, the function of this routine is 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 would be 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 before the Macintosh procedure `HLock` is called for any handle, to avoid heap fragmentation. 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 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}
```

```
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 or not 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.

```
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 length 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 when setting up their window. For example, a picker might make the following call:

```
myWindow := WindSetup (width, PickStdRows * DefaultListCellSize,
resType, dad^^.name);
```

```
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. Display of every alert should use this routine. The `which` parameter determines the kind of alert that is displayed.

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

```
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 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.

```
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).

```
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 as Template is selected, `MetaKeys` indicates that the Command and Shift 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 (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 such as New, Open, Open as Template, Open General, Close, Get Info, Save, Revert, Cut, Copy, Paste, Clear, and Duplicate. It is usually sufficient to call only this routine from `DoMenu`, although you may want to release any of the resources that were loaded.

```
FUNCTION PickStdRows: INTEGER;
```

This function returns the number of rows that should be displayed in a picker window. This value is obtained from the Preferences dialog box. It is guaranteed that the number of rows returned will all fit on the screen. A picker that doesn't use text rows for its display can determine the height in pixels with the following calculation:

```
height := PickStdRows * DefaultListCellSize;
```

```
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 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 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 ResEdit itself. 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);
```

Internal routines

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

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

`CallPBirth` starts a picker. This routine will rarely be used. It directly calls the `PickBirth` routine of the picker with the specified `id`.

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

`CallEBirth` starts an editor. This routine will rarely be used. It directly calls the `EditBirth` routine of the editor with the specified `id`. This routine is used by the `GiveEBirth` routine described earlier in this chapter.

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

`CallEvent` passes an event to the specified window. This routine will rarely if ever be used. If an event must be passed to a specific ResEdit window, the call would be made as follows:

```
CallEvent (evt, theWindow^.refCon, theWindow^.windowKind);
```

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

`CallMenu` passes a menu command to the specified window. This routine will rarely if ever be used. For example, if a Close command must be passed to a specific ResEdit window, the call would be made as follows:

```
CallMenu (fileMenu, closeItem, theWind^.refCon, theWind^.windowKind);
```

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

Given a handle `h` to a resource, `CopyRes` makes a copy of the resource to the resource file specified by `refNum`. Note that the handle is changed, so you can't keep track of your resource by saving its handle before using `CopyRes`. If `makeID` is TRUE, a unique ID will be assigned to the copy; otherwise, it retains the ID of the original. `CopyRes` returns a handle to the new copy (in the new file). This procedure is called from the `PickMenu` procedure described earlier.

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

`DoKeyScan` handles key-down events for pickers. The `offset` parameter is the byte offset into a cell where the string to match starts. This procedure is called from the `PickEvent` procedure described earlier.

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

`DoListEvt` is called from `PickEvent` and should normally not need to be called from elsewhere.

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

`DupPick` is called from `PickMenu` and should normally not need to be called from elsewhere.

```
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 of 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).

```
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.

```
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.

```
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.

Obsolete routines

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

```
PROCEDURE AppRes;
```

Use the `UseAppRes` procedure instead.

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

No longer supported.

```
FUNCTION Count1Res (t: ResType): INTEGER;
```

Use the `Count1Resource` toolbox procedure instead.

```
FUNCTION Count1Type: INTEGER;
```

Use the `Count1Types` toolbox procedure instead.

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

Use the `CheckError` procedure instead.

```
FUNCTION FileNewType types: ListHandle; VAR s: str255): BOOLEAN;
```

Use the `GetType` procedure instead.

```
PROCEDURE Get1IndxType (VAR theType: ResType; i: INTEGER);
```

Use the `Get1IndType` toolbox procedure instead.

```
FUNCTION GetResLoad: BOOLEAN;
```

No longer supported.

```
PROCEDURE MySeedFill (srcPtr,dstPtr: Ptr; srcRow,dstRow,height,words:
INTEGER; seedH,seedV: INTEGER);
```

No longer supported.

```
PROCEDURE ResEverest;
```

Use the `UseResFile` procedure instead.

```
FUNCTION RevertResource(h: Handle): Boolean;
```

Use the `RevertThisResource` procedure instead.

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

Use the `WindReturn` procedure instead.

Appendix A The 'KCHR' Resource

THIS APPENDIX CONTAINS MORE INFORMATION about the 'KCHR' resource, its structure and 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 keyboard-type-independent information about the key being depressed.

Exceptions to the rule

Some countries have different layouts for different keyboards, mostly for historic reasons. In order 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 to a Macintosh character set number based on the tables in the 'KCHR' resource. That character number, and also 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 Umlaut characters 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. Then it 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.)

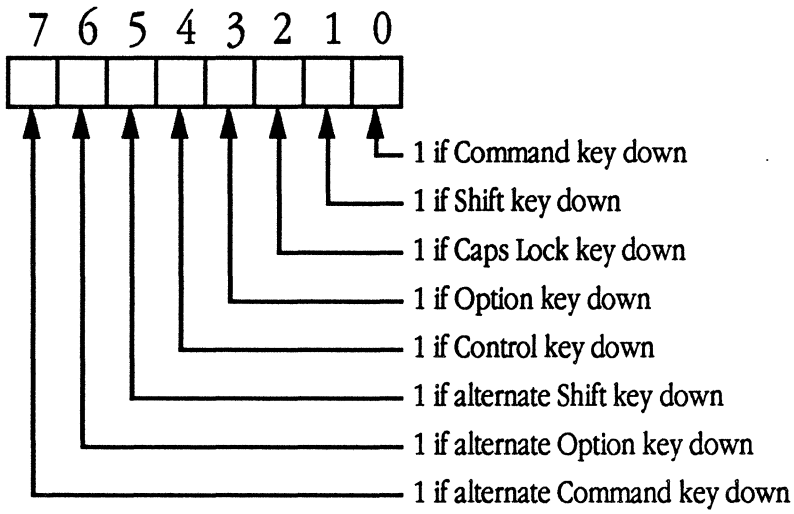
The structure of a 'KCHR' resource

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, since 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 Toobox 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 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 early 1989. 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! ■

Resource types defined for Rez and ResEdit

Type	Definition	Rez	ResEdit
actb	Alert Color Lookup Table	Types.r	Template
acur	?????	XXXXXXX	Template
ALRT	Alert Template	Types.r	Editor, Template
APPL	Application list (Desktop)	XXXXXXX	Template
BNDL	Bundle	Types.r	Template
cctb	Control Color Lookup Table	Types.r	Template
cicn	Color Icon	Types.r	XXXXXXX
clut	Generic Color Lookup Table	Types.r	Template
CMDO	For MPW Commando interface	Cmdo.r	XXXXXXX
cmnu	MacApp® temporary menu resource	XXXXXXX	Template
CNTL	Control template	Types.r	Template
crsr	Color Cursor	Types.r	XXXXXXX
CTY#	City list from MAP CDEV	XXXXXXX	Template
CURS	Cursor	Types.r	Editor
dctb	Dialog Color Lookup Table	Types.r	Template
DITL	Dialog Item List	Types.r	Editor, Template
DLOG	Dialog template	Types.r	Editor, Template
DRVR	Driver	SysTypes.r	Template
FBTN	MiniFinder button	XXXXXXX	Template
fctb	Font Color Lookup Table	Types.r	Template
FCMT	GetInfo comments from Desktop file	XXXXXXX	Template
FDIR	MiniFinder button directory ID	XXXXXXX	Template
finf	Font information	SysTypes.r	Template
FOND	Font Family description	SysTypes.r	Template
FONT	Font description	SysTypes.r	Editor, Template
FREF	File Reference	Types.r	Template
FRSV	ROM Font resources	XXXXXXX	Template
FWID	Font Width Table	SysTypes.r	Template
ICON	Icon	Types.r	Editor
ICN#	Icon and its mask	Types.r	Editor

(Continued)

Type	Definition	Rez	ResEdit
ictb	Color Dialog Item List (not handled yet)	XXXXXXX	XXXXXXX
insc	Installer Script	SysTypes.r	Template
INTL (0)	International Formatting information	SysTypes.r.	Editor...(same as itl0 but no longer used)
itl0	International Formatting information	SysTypes.r	Editor
INTL (1)	International Date/Time information	SysTypes.r	Editor...(same as itl1 but no longer used)
itl1	International Date/Time information	SysTypes.r	Editor
itl2	Intl Str Comparison Package hooks	SysTypes.r	XXXXXXX
itl4	International Tokenize	SysTypes.r	XXXXXXX
itlb	International Script Bundle	SysTypes.r	Template
itlc	International Configuration	SysTypes.r	Template
itlk	Intl exception dictionary for 'KCHR'	XXXXXXX	Template
KCAP	Physical layout of keyboard	SysTypes.r	XXXXXXX
KCHR	ASCII Mapping (software)	SysTypes.r	Editor
KMAP	Keyboard Mapping (hardware)	SysTypes.r	XXXXXXX
KSWP	Keyboard Swapping	SysTypes.r	XXXXXXX
LAYO	Finder's Layout resource	XXXXXXX	Template
MACS	Version # in System file	XXXXXXX	Template
MBAR	Menu Bar	Types.r	Template
mcky	Mouse Tracking	SysTypes.r	XXXXXXX
mctb	Menu Color Lookup Table	Types.r	Template
mcod	MacroMaker™ information	XXXXXXX	XXXXXXX
mdct	MacroMaker information	XXXXXXX	XXXXXXX
mem!	MacApp memory utilization	XXXXXXX	XXXXXXX
MENU	Menu	Types.r	Template
minf	Macro info (MacroMaker)	XXXXXXX	Template
mntb	MacApp Menu Table(relate cmd # to menu)	XXXXXXX	XXXXXXX
mppc	MPP Configuration resource	SysTypes.r	XXXXXXX
mxbc	Foregnd, backgnd colors for MacsBug	XXXXXXX	XXXXXXX
mxbi	Initial settings for MacsBug	XXXXXXX	XXXXXXX
mxbm	Macros for MacsBug	XXXXXXX	XXXXXXX
mxbt	Templates for MacsBug (byte count)	XXXXXXX	XXXXXXX
mxwt	Templates for MacsBug (word count)	XXXXXXX	XXXXXXX

(Continued)

Type	Definition	Rez	ResEdit
NFNT	Font description	SysTypes.r	XXXXXXX
nrct	Rectangle position list	SysTypes.r	Template
PAPA	????	XXXXXXX	Template
PAT	QuickDraw Pattern	Types.r	Editor
PAT#	QuickDraw Pattern List	Types.r	Editor
PICT	QuickDraw Picture	Types.r	Template
pltt	Color Palette	Types.r	Template
ppat	Pixel Pattern	Types.r	Template
ppt#	Array of ppats	XXXXXXX	XXXXXXX
PRC3	Print record (PREC) id = 3	XXXXXXX	Template
PSAP	????	XXXXXXX	Template
ROv#	ROM Resource Override	SysTypes.r	Template
scrn	Screen configuration	SysTypes.r	Template
seg!	MacApp item of some sort	XXXXXXX	XXXXXXX
SICN	Small Icon	Types.r	Editor
SIGN	????	XXXXXXX	Template
SIZE	MultiFinder Size information	Types.r	Template
snd	Sound	SysTypes.r	XXXXXXX
STR	PascalStyle String	Types.r	Template
STR#	PascalStyle String List	Types.r	Template
TEXT	Unlabeled string.(Same as minf)	XXXXXXX	Template
TMPL	ResEdit template	XXXXXXX	Template
vers	Version	SysTypes.r	Template
wctb	Window Color Lookup Table	Types.r	Template
WIND	Window template	Types.r	Editor, Template

Appendix C **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 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 many pictorial fonts available as bitmaps for dot-matrix printing as well. ■

Second digit \ First digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SPACE	0	@	P	`	p	Ä	ê	†	•	¿	–		
1	SCH	DC1	!	1	A	Q	a	q	Å	ë		±	¡	—		
2	STX	DC2	"	2	B	R	b	r	Ç	í	¢	£	¬	“		
3	ETX	DC3	#	3	C	S	c	s	É	ì	£		÷	”		
4	EOT	DC4	$	4	D	T	d	t	Ñ	î	§	¥	*f*	‘		
5	ENQ	NAK	%	5	E	U	e	u	Ö	ï		m	ª	’		
6	ACK	SYN	&	6	F	V	f	v	Ü	ñ	¶	d	D			
7	BEL	ETB	'	7	G	W	g	w	á	ó	ß	Â	•	‡		
8	BS	CAN	(	8	H	X	h	x	à	ò	®	'	‣	ÿ		
9	HT	EM	)	9	I	Y	i	y	â	ô	©	p	…			
A	LF	SUB	*	:	J	Z	j	z	ä	ö	™	Ú	␣			
B	VT	ESC	+	;	K	[	k	{	ã	õ	´	ª	À			
C	FF	FS	,	<	L	\	l	|	å	ú	¨	º	Ã			
D	CR	GS	-	=	M	]	m	}	ç	ù		W	Õ			
E	SO	RS	.	>	N	^	n	~	é	û	Æ	æ	Œ			
F	SI	US	/	?	O	_	o	DEL	è	ü	Ø	ø	œ			

␣ Stands for a nonbreaking space, the same width as a digit.

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

Index

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 the Apple LaserWriter® printers. Line art was created using Adobe Illustrator™. MacDraw® and MacPaint® were also used to create art for this manual. POSTSCRIPT®, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

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.

2/23/89