# MACINTOSH RESOURCES

## INTRODUCTION

This chapter gives a short description of what Macintosh resources are, how they are used, and why they are important. If you already know about resources, you will probably want to skip ahead to the chapter, "Editing Files", where the documentation of Resorcerer's operation begins.

For a complete description of resources on the Mac, see Apple's official documentation in *Inside Macintosh* (Addison-Wesley).

## FILES AND RESOURCES

Every computer system today lets its users keep various kinds of data in files. The data can represent anything: the text of a word-processing document; a set of tabulated numbers that a statistical analysis or spreadsheet program might read; or even the actual machine code of a program that, when run, causes the computer to read in and process the data from some other data file.

Typically, each file can have a name or icon by which the operating system recognizes the file, and the system makes it easy for the user to name, organize, and manipulate files as units of data, without regard to the form the data takes within the files.

The internal structure of each file, however, can be anything. Every file is nothing but a collection of bits whose interpretation is dependent only on who (e.g. which program) might read or write these patterns of 1's and 0's. Most computer hardware supports the grouping of the bits in a file into 8-bit bytes, or 16- or 32-bit quantities, but otherwise the interpretation of how the data is organized, what encoding it represents, etc., is entirely up to the programs that read and write the data (e.g., is it a text file, is it a bitmap picture file, etc.). The Mac file system distinguishes among files using four-character file type and file creator codes, but these are not part of the file data, and can be ignored by any program if it chooses to.

Over the years many standards have evolved that specify how the data in a particular kind of file should be interpreted by the programs that might read that file. But with the exception of some of these standards, such as ASCII text files, few are compatible with each other. In particular, program files that contain compiled machine code and data are nearly always machine (and usually operating system) specific. Unfortunately, there are as many different ways to store and organize the symbols, strings, and other data of a compiled piece of code as there are compilers.

The Macintosh computer file system solves part of this problem by extending the definition of a file, and by standardizing the way in which all Mac programs and their data files store various small chunks of information that are common to many different programs. These chunks are called a file's *resources*. A resource may be a piece of text (of which there are at least 5 different "standard" types), a list of strings or rectangles, menus, icons, fonts, color specifications, entire dialogs, pictures, or custom pieces of information a program might need.

# RESOURCES EXPLAINED

Every file in the Mac file system consists of two parts, one for data and one for resources. You can think of these two parts either as two files that are inseparably linked and always referred to using the same name and icon, or as one file with one name and icon that has two independent parts, or *forks*. The word *fork* is intended to convey the notion that neither of the two parts precedes the other; they exist in parallel.

The data fork of a Mac file contains plain data, and is entirely analogous to files on other computers: bytes in the data fork are accessed either sequentially or randomly from some given position. Other than that, the Mac assumes no other structure or interpretation for the data, leaving that up to the programs that read and write that data. Resorcerer can optionally treat the data fork of any file as if it were just another resource (for more on this, see the **Opening Files** and **Preferences** chapters later on).

The resource fork of a Mac file contains bytes just as other files do, but these bytes are grouped together into resources, which are individually addressable by *type*, resource *ID number*, and/or a *name*. The Macintosh Toolbox contains a set of special subroutines, called the Resource Manager, that are meant to be called by every program that runs on the Mac. These routines let programs create, retrieve, change, and store the resources of any

file's resource fork without having to worry about the exact manner in which the resources are stored. The Resource Manager maintains another data area, called the *resource map*, within the resource fork; the resource map is essentially the table of contents the Resource Manager needs to find the data and attributes for any resource in the file.

Each resource consists of a set of 0 or more bytes that is read and written as a single data structure between the disk file and a program's memory, for whatever purposes the program may require. Resources are internally stored in standard Macintosh Memory Manager memory blocks, accessed by handle pointers. Usually a copy of the resource is read in from the file, to be used once and then discarded; in other cases, resources are copied into memory where they stay throughout the execution of a program. Resources can also be used to record information about the state of a program, so that the user can restart the application at a later date and have it start up in the same state as it was the last time. In this case, if the information has changed, the application can write the resource back out when it quits.

One benefit of resources is that they can be used to divide the code of a large program up into smaller, more manageable pieces (often called "segments"). The same Mac Toolbox calls that your program uses to access an icon or a font are also used by the Mac operating system to manage how your program is loaded into memory and run. In fact, most typical Mac application files consist solely of a set of resources, some of which are code and some of which are data, with no information at all kept in the file's data fork.

Obviously different types of resources cannot be used interchangeably: for example, a resource that contains program code should not be used in place of a resource that contains the data for a picture (drawing the machine code would look like garbage, and executing the picture data would crash the Mac in a big hurry). Thus, resources need to be classified into types to distinguish the different data structures.

Fortunately, the Macintosh has a rich set of predefined resource types for most common resources. Each type has a documented (i.e. standardized) format that explains the purpose and interpretation of each byte in the resource. These predefined types help all programs on the Mac work similarly, which makes life easier for the user as well as the programmer.

Another important aspect of resources is that they allow the Mac's application programs to place a variety of non-code textual information into a well-structured and known place, the resource fork of the program file. Strings, text, menus, dialog items, etc. can then be quickly changed without having to recompile the program's source code. One benefit of this is that the text in a compiled Macintosh program that uses resources can be

translated easily into another language, making the program more usable in international markets.

Of course, the data in each resource is in a machine-readable, not human-readable, form. Thus we need special resource editing programs like Resorcerer to help us manipulate resources quickly and easily without having to worry too much about the internal bits and bytes.

# RESOURCE TYPES

Every resource in a Macintosh file has a 4-character resource type, indicating that the resource's data is in the required format for that type. This leaves enough room to create literally billions of types of resources, since 4 bytes can encode some 4 billion unique types. All extended 8-bit ASCII characters, including space characters, are significant when used as one of the 4 characters in the type. Uppercase and lowercase characters are considered different as well.

Examples of resource types:

'DLOG', 'MENU', 'CODE', 'STR ', 'PICT', 'sysz', 'dctb', 'View', 'DLGX'.

**Note:** Resource types with *all* lower-case letters are considered reserved for Apple. If you create your own resource type, it should have at least one upper case character in it, preferably the first character.

# RESOURCE IDS

Resources of the same type can be distinguished by their resource ID, which is a 16-bit signed integer from -32768 to 32767. All resources of a given type in any one file should have unique IDs. Otherwise, if a program requests one of two resources of the same type that have the same ID, the Resource Manager delivers one or the other of them in an undefined order.

All IDs less than 0 are usually considered reserved for Apple's use, and IDs from 0 to 127 are generally off-limits as well, being reserved for drivers, desk accessories, and other special system resources, particularly ones that can own resources (see below). Standard application resources should usually have IDs greater than or equal to 128.

# RESOURCE NAMES

Every resource can have an optional resource name of up to 255 characters. The optional name is kept internally as a Pascal string along with the resource's type, ID, attributes (see below), and data in the resource fork of the file.

Optional names help distinguish resources in the Resource List and are highly recommended, since humans (including some programmers) read English better than they read numbers. A program can ask the Resource Manager to retrieve resources by type and name rather than by type and ID. Note that when doing so, the Resource Manager ignores any case differences among names (although diacritical marks do make a difference).

# RESOURCE ATTRIBUTES

Associated with every resource is a set of 6 bits that code for the presence or absence of certain attributes. These bits govern the way the Mac treats the resource in common situations that occur while a program is running.

**The Locked bit**

With the Locked bit set, the resource data will be locked in memory after it is initially read in from its file. This attribute will override the Purgeable attribute.

**The Purgeable bit**

With the Purgeable bit set, the resource data is set to be purgeable at any time after it has been read into memory. Many resources, especially templates of various kinds, can be set to be purgeable, if they are only needed for a short time. The Locked attribute will override the Purgeable attribute.

**The Preload bit**

With the Preload bit set, a resource's data is read in automatically when its file is initially opened; otherwise the resource waits on disk until a program specifically requests it.

**The System Heap bit**

Normally, resources are read in and placed in the application's heap; with the System Heap bit set, however, the resource will be installed in the System heap instead. This bit should not usually be set for an application's resources.

**The Protected bit**

With the Protected bit set, a resource's ID, name, attributes, and data cannot be changed by the application until the Protected bit is cleared.

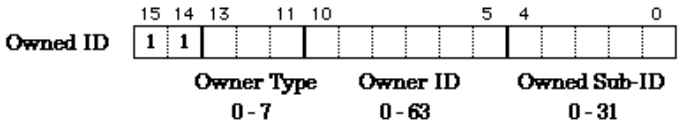
**The Extension bit**

The Extension bit is currently a private Apple resource attribute that is intended to allow Apple to extend the Resource Manager's capabilities in the future. The bit is primarily used to signify that a resource's data has been compressed using a private Apple algorithm, and that the Resource Manager should automatically decompress the data before delivering it to any caller. Decompression only occurs if the application is running under System 7 or higher. As of this writing (System 7.1), the Resource Manager cannot compress resources whose Extension bit has been set.

# OWNED RESOURCES

Certain system resource types are said to explicitly *own* other resources if the owned resources have special IDs that encode their relationship to the owning resource. For instance, a Desk Accessory resource may always need a collection of resources to call upon to display a dialog, string, or picture. Since the Desk Accessory is a 'DRVR' resource, it should always be kept together with the resources it needs. By giving these resources special IDs that, in effect, declare the 'DRVR' to be their owner, various system utility programs, such as Font/DA Mover or Resorcerer, can keep the whole group together.

The resource types that can own other resources in this way are 'DRVR', 'WDEF', 'MDEF', 'CDEF', 'PDEF', and 'PACK' resources (two other types have been reserved for future use, and are known as 'RSV1' and 'RSV2').




Resource IDs whose 2 most-significant bits (bits 15 - 14) are set are reserved for owned resources. Since the most-significant bit is the sign bit, these IDs are always negative.

Following these 2 bits, a 3-bit field (bits 13 - 11) specifies one of the 8 possible resource types that the owning resource can have:

| | | |
|---|---|---|
| 000 | DRVR | Driver or Desk Accessory |
| 001 | WDEF | Window Definition Function |
| 010 | MDEF | Menu Definition Function |
| 011 | CDEF | Control Definition Function |
| 100 | PDEF | Printer Definition Funtion |
| 101 | PACK | Package |
| 110 | RSV1 | Reserved |
| 111 | RSV2 | Reserved |

After the 3-bit owner type field, a 6-bit field (bits 10 - 5) specifies the owning resource's ID, (which can thus only be in the range 0 - 63).

The final 5-bit field (bits 4 - 0) of the 16-bit ID number is used to distinguish among the 32 possible resources of the given type that can be owned by the given owner. It is often called the sub-ID field, since it serves the same function as the entire resource ID of a non-owned resource.

# RESOURCE FILE LIMITATIONS

As of this writing (System 7.1), a Macintosh resource file cannot contain more than 16 megabytes of data, and any resource file cannot have more than 2727 individual resources. These restrictions are due to the Resource Manager's internal design, which relies on 24- and 16-bit offsets.

As a rule of thumb, you should not keep more than 500 resources of any given type in a single file. This is because the Resource Manager searches for resources linearly. Resource files should not be treated as full-scale application databases.