CONFIGURING RESORCERER

INTRODUCTION

This chapter explains how you can configure various aspects of Resorcerer's operation. Future versions of Resorcerer may provide a more direct means of manipulating these settings. Make sure that the current **Preferences...** dialog doesn't already support the settings you are interested in.

You do not need to do anything described in this chapter to run Resorcerer successfully; however, you may benefit from being familiar with some of these configuration parameters, especially those that pertain to future compatibility or current Mac toolbox bugs.

You will need to use Resorcerer to edit a copy of itself in order to set various of its configuration parameters, most of which are kept in Resorcerer's own resources. You should be very familiar with the general aspects of editing resources with Resorcerer, especially using the Hex, Menu, and StringList Editors.

TOPICS COVERED

- Maximum number of files
- Scrap types to import
- Resource type descriptions
- Finder file flags
- Resource map flags
- The read-only resource map bug
- Screen copying timeout
- The 'ictb' Dialog Manager bug
- Adding custom screen sizes
- Displaying resource data

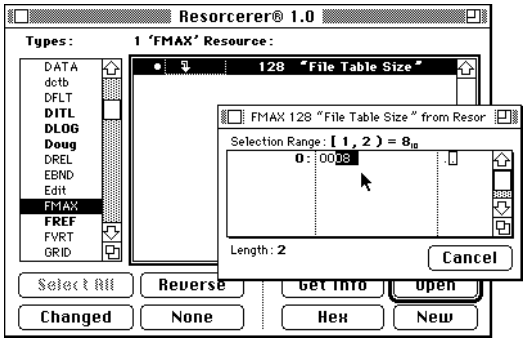
MAXIMUM NUMBER OF FILES

When Resorcerer first starts up it allocates a file table with a maximum number of entries. One entry is used for every simultaneously open file that you may want. Initially, the size of the file table is 16; if you try to open more, Resorcerer will complain about the table being full.

If you need to edit more files simultaneously than the current maximum allows, you must change the startup file table size. This number is kept in the first two-byte word of one of Resorcerer's own resources, 'FMAX' 128.

Note: Currently, the size of each file table entry is about 1900 bytes, so you may not want to set the maximum number of files to a very high number indiscriminately. Resorcerer will ignore any file table count less than 2. Note also that the <Resource Scrap> file is not included in the file table.

To change the file table maximum, make a temporary copy of your working version of Resorcerer using the Finder's **Duplicate** command. Run the copy and open the original to read in all of the original's resources. Select 'FMAX' in the Types List, and then open the 'FMAX' 128 resource with the Hex Editor.



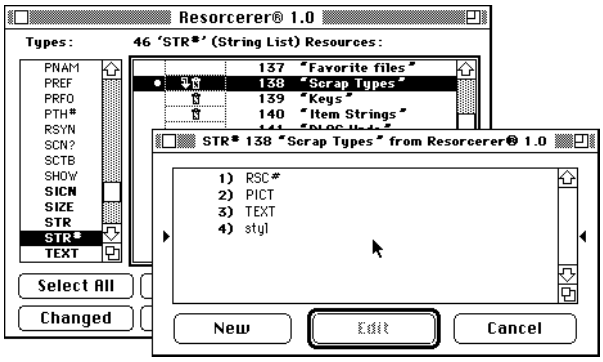
Byte 0 is the high-order byte of the file table count and should usually be 0, byte 1 is the low order byte of the count and it should contain the maximum number of files. Select byte 1 of the resource and type in the two hex digits for your new file count. Then close the resource, close Resorcerer's resource file, and quit the copy of the old version you are running.

Once you've saved the resource and closed the file, you should run the original you just modified in order to ensure that you've made the correct change. You can then throw the copy of Resorcerer you just made into the Trash, or keep it in your backup archives.

SCRAP TYPES TO IMPORT

Resorcerer's <Resource Scrap> is designed to import any type of scrap from the Mac clipboard, where the data is assigned the next free resource ID and given the name, "(from Desk Scrap)". Once this data is in the <Resource Scrap> file, you can then paste these new resources into the file you are editing.

The list of types that Resorcerer looks for in the scrap is kept in one of Resorcerer's own string list resources: 'STR#' 138 (the ID may vary in different versions, so you might want to search for the word "Scrap" in its name, since it is named "Scrap Types"). Resorcerer interprets the first four characters of each string in this resource as a scrap type name. If the Mac's clipboard file contains a scrap of any of these types, Resorcerer imports it into its <Resource Scrap> file, converting the scrap data to a resource.



To add a new scrap type or to remove an old scrap type from this list, make a temporary copy of your working version of Resorcerer using the Finder's **Duplicate** command. Run the copy and open the original to read in all of the original's resources. Select 'STR#' in the Types List, then open the 'STR#' 138 resource with the StringList Editor. Use the StringList Editor to make your changes.

The first string in the resource should be the type, 'RSC#', which is a special scrap type that Resorcerer recognizes as containing a list of resources, including resource data, names, and attributes for each resource in the list. The 'RSC#' type lets Resorcerer import multiple resources from another application that supports such a scrap type. When Resorcerer imports a piece of scrap of this type, it parses the single piece of scrap into all the separate resources it contains, and adds them to the <Resource Scrap> file.

The structure of a 'RSC#' scrap is described by the following template ('TMPL') fields:

OCNT	Resources	
LSTC	•••••	
	FLNG	Reserved (must be 0)
	TNAM	Resource type
	DWRD	Resource ID
	FBYT	Filler
	HBYT	Resource attributes
	ESTR	Resource name
	LHEX	Long byte count, followed by resource data
	AWRD	Align
LSTE	•••••	

Once you've saved the 'STR#' 138 resource and closed the file, you should run the original you just modified in order to ensure that you've made the correct change. You can then throw the copy of Resorcerer you just made into the Trash, or keep it in your backup archives.

RESOURCE TYPE DESCRIPTIONS

Each time you click on a type in a file's Types List, Resorcerer fills the Resources List with all resource of that type. It also redraws the label above the list. Because resource types can only be four characters, they tend to be cryptic abbreviations that are hard to remember.

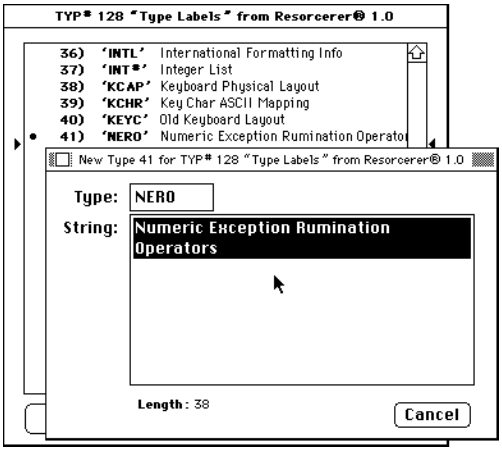
Therefore, when Resorcerer creates the new label above the list, it looks up a plain-language, unabbreviated string that gives a better idea of what the resource is all about. These etymological labels, and the types they belong to, are kept in a resource in Resorcerer's own resources: 'TYP#' 128, with the resource name, "Type Labels".

The TypeList ('TYP#') resource is very similar in structure to a standard StringList ('STR#) or Template ('TMPL') resource, and in fact the same Editor edits all three types internally. It is simply a list of four-character types with a Pascal string associated with each type.

As new resource types get defined and eventually standardized, or if you have private resource types whose function you want to be reminded of each time you browse them, you may want to add more label strings to this dictionary.

To do this, make a temporary copy of your working version of Resorcerer

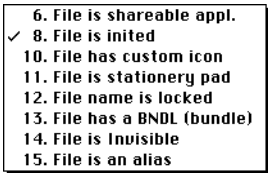
using the Finder's **Duplicate** command. Run the copy and open the original to read in all of the original's resources. Select 'TYP#' in the left list, and then open the 'TYP#' 128 resource with its Editor. Find an appropriate spot to insert your new type and its plain language string, and put it there.



Once you've saved the resource and closed the file, you should run the original you just modified in order to ensure that you've made the correct change. You can then throw the copy of Resorcerer you just made into the Trash, or keep it in your backup archives.

FINDER FILE FLAGS

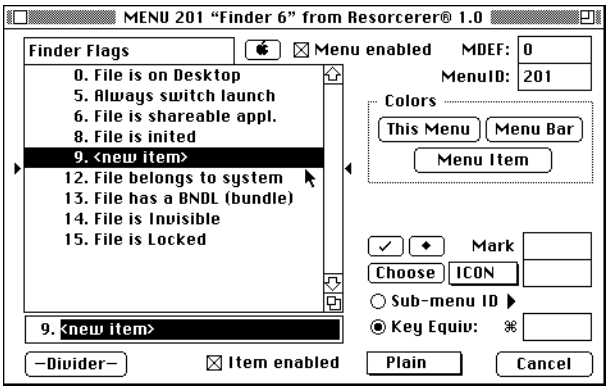
Each file on the Mac has a word of 16 attribute bits that help the Finder make decisions about the file in various situations. As the Mac System and Finder have evolved over the years, these bits have changed their meaning. Certain reserved bits in the past have been given new meanings; in other cases, some older meanings have been discontinued or redefined.



Resorcerer's File Window makes the values of certain of these bits visible to you in the **Finder Flags** pop-up menu that you can see when you choose **File Info** from the **File** menu. You can check or uncheck any item in the pop-up independently of any other item. Since each item in the menu explicitly starts with the bit number it represents, Resorcerer creates a new attribute word by scanning the pop-up for items that are checked or not checked, looking up the bit number at the start of the item, and setting or clearing that bit in the attribute word.

Because some of these bits have changed between System 6 and System 7, Resorcerer contains a menu for when you are running under System 6, and another menu for later systems. In either case, if you want to be able to

directly set a bit that is not in the menu because of some new feature that Apple has implemented, you should edit one or the other of these 'MENU' resources.



Make a temporary copy of your working version of Resorcerer using the Finder's **Duplicate** command. Run the copy and open the original to read in all of the original's resources. Select 'MENU' in the left list, and then open the 'MENU' resource for either popup with its Editor. After making the Resources List the Active List, you can search for the menu you want by typing "Finder" and tapping the TAB key to get to the right one.

Each item in the menu begins with the bit number of the attribute the item represents, followed by a '.' (period) and a space, followed by the item text that gives a hint about the bit's meaning. If the bit number is less than 10, you should precede it with an Option-Space, to make all the numbers line up vertically in the menu. Find an appropriate spot to insert your new bit number and label, and put it there. Since the bit numbers are explicitly stored within each item, the order of the items in the menu is immaterial. The 16 bits are numbered from 15 (high order bit) down to 0 (low order bit).

Note: The Finder Flags word uses bit numbers 3, 2, and 1 to encode one of 8 colors/labels with which the file's black & white icon might be marked. These bits are already taken care of in the **Color** pop-up menu elsewhere in the File Window dialog.

Once you've saved the resource and closed the file, you should run the original you just modified in order to ensure that you've made the correct change. You can then throw the copy of Resorcerer you just made into the Trash, or keep it in your backup archives.

RESOURCE MAP FLAGS

In addition to the file attributes of interest to the Finder, the resource fork of any file can have its own set of attribute flags. These flags affect how the Resource Manager behaves with respect to resources in your file. When an application calls the Resource Manager's `GetResFileAttrs` routine, the routine delivers a word some of whose bits encode various attributes that the resource map has. The Manager's `SetResFileAttrs` routine lets your program set the attributes.

In the same manner as explained in the previous "Finder File Attributes" section, Resorcerer enables you to set or clear certain of these bits according to the entries found in the **Map Flags** pop-up menu in the File Window. Because each item in the menu explicitly contains the bit number that the item represents, you can add, remove, or rearrange items in this menu using Resorcerer's Menu Editor. Resorcerer can only set or clear attribute bits for which there are explicit entries in the Map Flags pop-up menu; all other bits in the attributes word are left untouched.

Make a temporary copy of your working version of Resorcerer using the Finder's **Duplicate** command. Run the copy and open the original to read in all of the original's resources. Select 'MENU' in the Types List, and then open the pop-up 'MENU' resource with its Editor. After making the Resources List the Active List, you can search for the menu you want by typing "Map" and tapping the TAB key to get to the right one.

Each item in the menu begins with the bit number of the attribute the item represents, followed by a '.' (period) and a space, followed by the item text that gives a hint about the bit's meaning. If the bit number is less than 10, you should precede it with an Option-Space, to make all the numbers line up vertically in the menu. Find an appropriate spot to insert your new bit number and label, and put it there. Since the bit numbers are explicitly stored within each item, the order of the items in the menu is immaterial. The 16 bits are numbered from 15 (high order bit) down to 0 (low order bit).

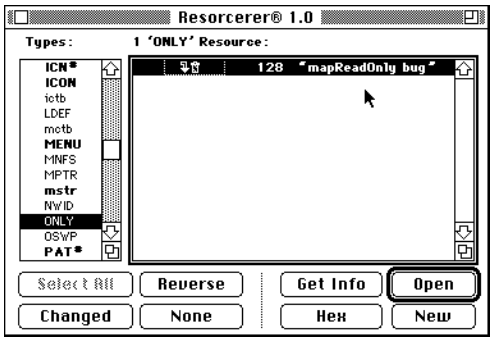
Note: Currently, most of the resource map attribute bits remain undocumented, reserved, and private to Apple's software, so you should only make these changes if you really, really know what you are doing. Anything Apple has not officially documented should always be considered as subject to change.

Note: The next section contains important information related to the attributes word bit ordering.

THE READ ONLY RESOURCE MAP BUG

There is a bug (or feature, depending on your point of view) in current versions of the Resource Manager that keeps it from being able to properly set one of the more useful documented resource map attributes, the `mapReadOnly` bit. This attribute bit adds an extra level of protection when any other application, via the Resource Manager, accesses your file.

Consequently, Resorcerer changes the resource map attributes directly in your file, bypassing the Resource Manager entirely after your file has been closed. This is, however, a violation of the resource map's privacy as a data structure, which only the Resource Manager should know anything about. If in the future Apple changes the resource map structure in such a way that the position of the map attributes word changes, then Resorcerer's current method will not work and more than likely will damage the file's map.



Because of this possibility, we have isolated the code that bypasses the Resource Manager into a small resource, 'ONLY' 128. If this resource is present, Resorcerer calls upon it to read and write only the resource map attribute word. If you delete this resource, or change its type or ID to anything else, Resorcerer will only attempt to set the resource map attribute word using the Resource Manager's `SetResFileAttrs` routine.

Sorcery: Although Apple has allocated a full 16-bit word to hold attribute bits within every resource map, the Resource Manager's `SetResFileAttrs` takes the low order byte of its word argument and places it in the *high* order byte of the word in the map. The bits in the low order byte of the resource map word remain completely reserved and private to Apple. Therefore, before the 'ONLY' resource delivers the map attribute word to Resorcerer, it swaps the high and low order bytes. When the attributes are given back to the 'ONLY' routine, it unswaps the bytes and saves them in the map. Thus, if you are privy to the meaning of these undocumented map attributes, you can configure your copy of Resorcerer to give you access to them by creating entries for bits 8 to 15 in the **Map Flags** pop-up menu described earlier.

If you are not privy to the meaning of these bits, changing any of them is guaranteed to break something (you should let sleeping dogcows lie).

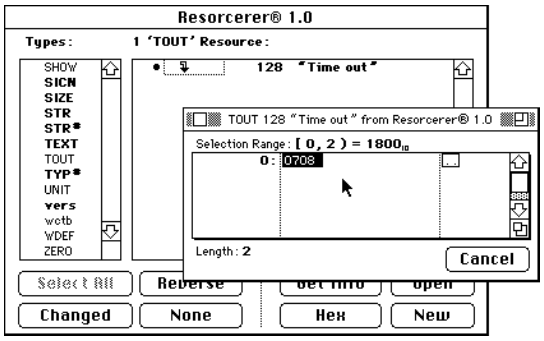
SCREEN COPYING TIMEOUT

Resorcerer's screen-copying facility uses a "marching ants" marqee that you can drag anywhere on your screen. Because Resorcerer is temporarily drawing and erasing screen real-estate that doesn't belong to it, and because your next mouse click is almost certainly positioned over some other application's window, you can't switch to another application while in this mode. Resorcerer is "MultiFinder-hostile" while you are screen-copying.

Since you do not need to actively do anything to remain in screen copy mode (i.e. you don't have to hold the mouse button), the possibility exists that you might get interrupted, leaving Resorcerer in a state where it is not allowing MultiFinder to give system time to other applications, which might need time to do important things.

Consequently, screen copy mode halts by itself after a timeout period. Initially, this timeout is set to 1800 ticks, or 3 minutes, but you may want to shorten it if this is too long. The timeout value is kept in Resorcerer's 'TOUT' 128 resource.

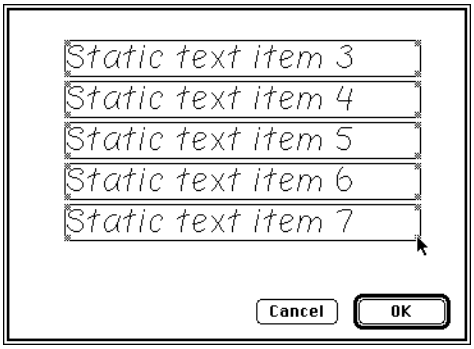
To change the timeout, use a copy of Resorcerer to edit the original. Open the original to read in all of the original's resources. Select 'TOUT' in the Types List, and then open the 'TOUT' 128 resource with the Hex Editor.



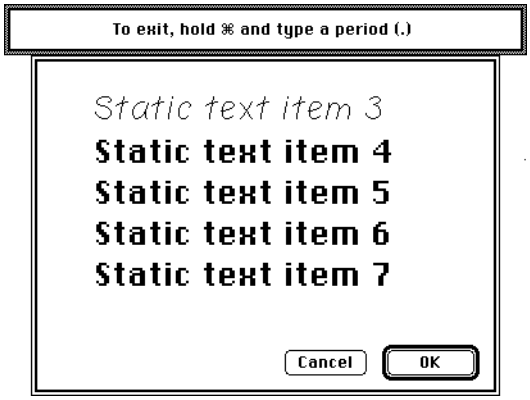
The length of this timeout period, in 1/60 second ticks, is kept in the first two-byte word in Resorcerer's 'TOUT' 128 resource.

Once you've saved the 'TOUT' resource and closed the file, you should run the original you just modified in order to ensure that you've made the correct change. You can then throw the copy of Resorcerer you just made into the Trash, or keep it in your backup archives.

THE 'ICTB' DIALOG MANAGER BUG



There is a bug in the Mac's Dialog Manager that keeps it from properly setting the font name and size for text items that refer to the *same* entry in its item color table ('ictb') resource. This is quite often the case since it's a good idea to make all text items in a dialog have the same style.



A typical symptom of the bug is that the first text item to have the custom style looks fine, but subsequent text items, although they keep their custom font size, revert to the System font (this bug was discovered with Resorcerer's **Try out...** command in the Dialog Editor).

The bug does not arise, however, if the 'ictb' resource is built with a *duplicated* (rather than shared) entry for each text item and its font name. Unfortunately, when a dialog has a lot of text items in it, these duplicated entries can add up to a lot of wasted space, which the structure of the 'ictb' resource was designed to avoid.

The Dialog Editor is configured to build an expanded 'ictb' resource (that is, one with unique text style entries per dialog item) when you save (or **Try Out...**) the dialog to which the 'ictb' refers. If, on the other hand, you hold the Option key down as you save (or **Try Out...**) your dialog, the Editor will create a compact 'ictb' resource with shared entries for the similarly styled or colored items in the dialog (this is how the above illustration was created).

If and when Apple fixes this bug, you can re-configure the Editor to swap the sense of the Option key so that the Editor will always create the compact 'ictb' resource *unless* you are holding the Option key down. To do this, you need to remove the 'OSWP' 128 resource ("Option Swap") from Resorcerer's own resources if there is one already there.

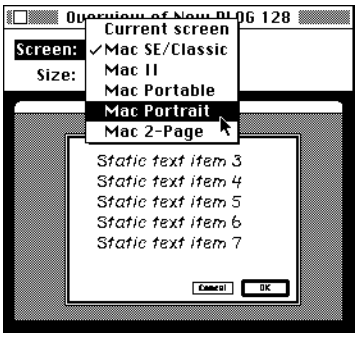
As usual, make a temporary copy of your working version of Resorcerer using the Finder's **Duplicate** command. Run the copy and open the original to read in all of the original's resources. Click on the **New** button while the Types List is active, and enter 'OSWP' as the new resource type, and 128 as the new 'OSWP' resource's ID. Click the **Create** button to create the resource using the Hex Editor.

You don't actually need to keep any data in the 'OSWP' resource. The Dialog Editor uses the resource's presence or absence to determine the sense to assign the Option key. If 'OSWP' 128 is there, the Editor defaults to creating expanded 'ictb's unless you hold the Option key down; when it is missing, the Editor creates compact 'ictb's unless the Option key is held down.

After you've added or deleted the 'OSWP' resource and closed the file, you should run the original you just modified in order to ensure that you've made the correct change. You can then throw the copy of Resorcerer you just made into the Trash, or keep it in your backup archives.

Once the Dialog Manager bug gets fixed (this is very unlikely, however), you should re-open Resorcerer and delete the added 'OSWP' resource, so that compact 'ictb's will once again be created without intervention on your part.

ADDING CUSTOM SCREEN SIZES



The Dialog Editor supports an Overview Window that displays your dialog or alert on a half-size replica of a Mac screen. You can change to different screen sizes by using a popup menu to choose from among the various standard Macintosh screens that Apple manufactures.

If you want to add entries to this popup menu, make a copy of Resorcerer, double-click on the copy, and open the original. Use the **Find All...** command to find and open the menu that contains the string "Classic" or whatever (as of this writing, it is 'MENU' 205).

After adding the names of your custom screens to the popup menu, close the 'MENU' resource and open the 'SCTB' 128 resource. The format of this resource is simply an indefinite list of pairs of 2-byte words. If you want to edit the resource with the Data Editor, create a template like this:

```
LSTB
        DWRD Screen height
        DWRD Screen width
LSTE
```

The entries in this list must parallel the entries in the popup menu. If you have appended two screen names to the menu, you need to append two (height, width) coordinate pairs to the 'SCTB' resource.

Close the resources, and then close the file. Run the original copy of Resorcerer to make sure it's working with the new screen sizes.

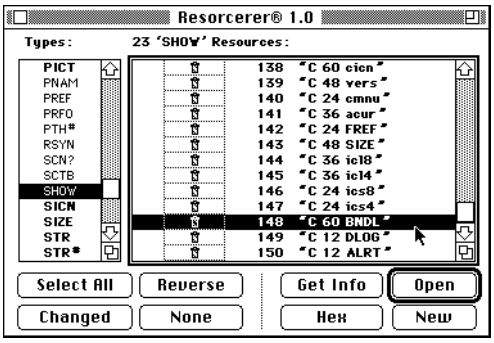
DISPLAYING RESOURCE DATA

Each time you click on a type in the Types List, Resorcerer fills the Resources List with all the resources of that type. The default entry in this list consists of the resource attributes, the resource ID, and its optional name string, but there is no indication of what the value of the resource's data might be.

It is, however, nice to be able to see the value of the resource data to get a better idea of which resource is which. This is as true for resources whose data is graphical in nature, such as icons, cursors, pictures, etc., as it is for strings, or file references, versions, or other "non-graphical" data.

Resorcerer supports an extensible mechanism for viewing resource values in the Resources List of the File Window. This capability is implemented by adding resources of type 'SHOW' to Resorcerer's resources.

Each 'SHOW' resource contains compiled code for a simple, single entry-point subroutine that takes one argument, a pointer to a `ShowRecord`. The `ShowRecord` contains fields such as the handle to the resource data, its type and attributes, important pre-computed values, etc., that the subroutine uses to display the resource value. Whenever Resorcerer's List Manager needs to display the value of a given resource that has a 'SHOW' resource installed for it, the List Manager loads and calls the code resource to do so.



The resource name of each 'SHOW' resource declares to Resorcerer the calling convention to use, how high the list cell should be, and what type of resource the subroutine knows how to display. The first character of the name should be either 'C' or 'P', to indicate whether the compiled code is a C subroutine or a Pascal subroutine. This lets Pascal programmers work in Pascal, and C programmers work in C, without forcing either camp to follow the conventions of the other. The second character in the name should be a space, and is ignored. The third character begins the digits of the height of the Resources List cell in pixels that Resorcerer should use. Another space follows the last digit, and the next four characters of the name indicate which type of resource the 'SHOW' resource displays.

Example: 'SHOW' 741, with the resource name, "C 36 SHNK", would get called as a C subroutine each time Resorcerer wanted an 'SHNK' resource drawn in some Resources List cell. Prior to displaying any 'SHNK' resources, Resorcerer would build the Resources List with a cell height of at least 36 pixels.

Note: The cell height for 'SHOW' resources should be a multiple of 12 pixels, and should not be more than 96 pixels. Also, the code within the 'SHOW' resource should not assume that the cell height is in fact the same as in its resource name, only that the cell height is at least the value in the name. The actual cell height is available by looking in the `ShowRecord`.

DESIGN CONSIDERATIONS

The `ShowRecord` contains a field, `message`, whose value selects what function the 'SHOW' resource should perform. Currently, only two messages are defined:

```
#define shDeclareVersion   0
#define shDrawResource     1
```

The `shDeclareVersion` message is sent to the 'SHOW' resource only once each time the Resources List is built for a given resource type. It is a request to tell Resorcerer what version the 'SHOW' resource code supports. Your 'SHOW' should set this to 0, which is the version being described here. No other version is currently defined.

Resorcerer's List Manager takes care of highlighting the contents of list cells, and of drawing the resource attributes, resource ID, and any owner decoding of the ID. After calling the 'SHOW' resource to draw the data, Resorcerer draws the resource name at the current pen position. Currently, the resource attributes and ID always appear in the left part of the list cell.

It is your 'SHOW' resource's responsibility to determine what part of the resource to draw and where to draw it within the list cell bounds. It must also leave the QuickDraw pen at the start of the position where Resorcerer can finish drawing the resource name. This represents a compromise between the pure generality of allowing the 'SHOW' to draw everything (e.g. a complete List Definition Function ('LDEF')), and uniformity of interface which makes the program easier to use at the expense of the fun of programmers. It also allows 'SHOW' resources to

be somewhat smaller than 'LDEF's, and makes them easier to write.

If we need more general 'SHOW's in the future, we'll create a new class of them with different versions. Note that if a 'SHOW' code resource does absolutely nothing (e.g., it just returns), then the list cell will look like a standard default Resources List cell.

STRUCTURE OF A SHOWRECORD

The following declaration is taken from a SHOW.h C header file:

```
#include <Windows.h>

// Pascal or C function to deliver another resource from same
file

typedef struct {
  Handle (*GetResData)(ResType type, short id);
  } C_Callbacks;

typedef struct {
  pascal Handle (*GetResData)(ResType type, short id);
  } P_Callbacks;

typedef struct {   // Version 0 ShowRecord structure

  long version;    // Version of SHOW reported back to
Resorcerer
  short message;   // Function selector
  short errCode;   // Place to tell caller bad news

  unsigned short flags;  // Reserved flags bits
  Rect *bounds;          // Current List32Manager's cell
bounds
  GrafPtr port;          // Current port cell is being drawn
in
  short attrBaseline;    // Attribute box's baseline (Y)
  short leftMargin;             // Where data can begin being
drawn

  short sysFontSize;            // Current system font size
  short reserved;

  short fontNum;   // Current font (usually Geneva 9 bold)
  short fontSize;  // Current font size
  short fontFace;  // Current font face
  FontInfo fInfo;  // Current font ascent, descent, etc.
```

```
  Handle hndl;     // Handle to unlocked detached resource
data
  ResType type;    // This resource type (4 characters)
  short ID;        // This resource ID
  short attrs;     // Resource attributes
  long size;             // Resource size in bytes

  Byte *IDstr;     // Pascal str of resource ID (6 chars max)
  Byte *ownerStr;  // Pascal str of Owner breakdown in [ ]'s
  Byte *sizeStr;   // Pascal str of resource size
  Byte *nameStr;   // Pascal str of resource name

  void *rsrvd1;    // Private Resorcerer internal stuff */
  void *rsrvd2;

  union {
     C_Callbacks C;
     P_Callbacks P;
     } call;

  #define NUMSHOWEXTRA   2
  long extra[NUMSHOWEXTRA];           // For expansion; zero
if unused

  } ShowRecord, *ShowRecordPtr;
```

Each 'SHOW' resource should be compiled as a single-entry-point C subroutine or Pascal procedure that takes a pointer to a ShowRecord as its only argument. With the exception of the version and errCode fields, you should not write to any of the fields in this structure.

The version field is where the 'SHOW' resource tells Resorcerer what version it is when the message field is shDeclareVersion. The type of ShowRecord that Resorcerer provides during later calls may depend on the value returned in the version field. You should set this to 0. All 'SHOW' code resources must support the shDeclareVersion message.

message contains an integer selector that indicates what function Resorcerer wants the 'SHOW' resource to perform. Currently, only shDeclareVersion and shDrawResource are passed in this field.

The errCode field is set to 0 prior to the call, so if all goes well you can ignore errCode. Currently, no errors are defined, so you can just leave it alone. However, you might want to give the user an indication of

some error condition by drawing something into the list cell. For example, the 'SHOW' resource for animated cursor ('acur') resources looks up the cursor IDs in the data, and attempts to draw all the cursors in the list cell. If it finds that the 'acur' resource refers to a non-existent 'CURS' resource, then it indicates this by drawing a gray square where the cursor would normally appear.

The `flags` field is currently reserved.

`bounds` is a pointer to a rectangle whose coordinates are the current list cell Resorcerer wants the resource drawn in. These are window coordinates. Resorcerer's List Manager has already set the clipping region to this rectangle.

`port` is the current window's port.

The `attrBaseline` field is the Y value of the baseline that matches where Resorcerer has just drawn the boxed resource attribute icons. If you want any resource name to appear to the right of the data you've drawn, you can use this value to place the Quickdraw pen at the proper Y position.

The `leftMargin` field is the X value in window coordinates at which it is okay to begin drawing resource data. Area to the left of this position has been used by Resorcerer for the attributes, ID, etc.

The `sysFontSize` field is made available here so you don't have to figure it out. If you want to display any string data from your resource in the current system font, you will want to use the value of this field while you temporarily set the window font to the system font to draw the string.

The font number, size, and face of the font that Resorcerer is currently using to display the resource ID and name are to be found in the `fontNum`, `fontSize`, `fontFace` fields. The `fInfo` field contains this style's font information record. You should restore the current text style to these values if you change them while you draw the data.

The `hndl` field is a handle to the unlocked, non-purgeable, detached resource data. It is not a handle that the Resource Manager will recognize as a resource. You will probably need to lock it while you scan or draw the data in it. When you've finished, you should restore any changed state to whatever it was before you changed it.

The `type`, `ID`, `attrs`, and fields provide the resource's type, ID,

attributes, and size. Since the data in `hndl` is a detached resource, these fields provide the pertinent resource information that you would otherwise not be able to determine. You can ignore them for now.

The `IDstr`, `ownerStr`, `sizeStr`, and `nameStr` fields are pointers to Pascal strings that contain text versions of the resource's ID, decoded owner ID, size, and name. With the exception of `nameStr`, you can ignore these fields, since Resorcerer takes care of drawing them. Resorcerer also draws the value of nameStr; however, the position at which it begins to draw the name is up to you, and may depend on whether the string is empty or not. For instance, when the value of a resource, such as a pattern list ('PAT#'), extends arbitrarily far to the right, you will want to leave room in the bottom of the list cell for Resorcerer to fill in the resource name, starting at `leftMargin`. If the resource name is empty, however, this looks unbalanced, and in this case you might want simply to center the resource value vertically in the list cell.

`call.C.GetResData or call.P.GetResData` is a pointer to a Resorcerer function that delivers a handle to a detached *copy* of any resource in the same file as the resource being drawn. It is either a C or Pascal function, depending on the first character of the 'SHOW' code resource's name, and it takes two arguments: the four-byte type and two-byte ID of the resource you want to peek at. You must call `GetResData` to access the latest edited value of any given resource; calling `Get1Resource` or any other Resource Manager routine will only deliver the resource as it was last saved on disk, or fail altogether. Since the handle returned is a copy, it is your 'SHOW' routine's responsibility to dispose of it (using `DisposeHandle`) before returning. `GetResData` delivers `NIL` if it can't find the resource, or has any other problem.

`reserved`, `rsrvd0`, `rsrvd1`, and `rsrvd2` are private Resorcerer fields.

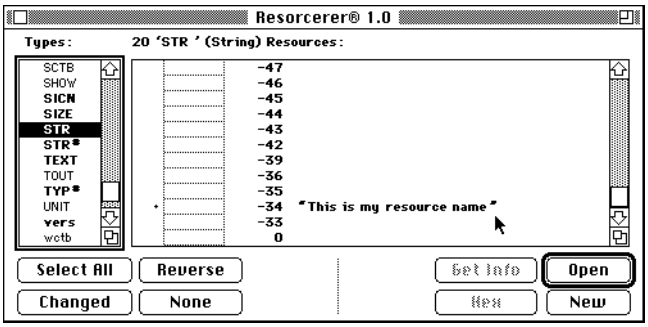
The `extra` array is for future expansion, and all values in it are currently set to 0.

EXAMPLE OF A 'SHOW' CODE RESOURCE

The following is an example of the C code you might use to create a 'SHOW' "C 24 STR " resource (don't forget the trailing space character) that displays the value of a Pascal string resource in a list cell at least 24 pixels high. Regardless of the current window font, or current list font, we choose to draw all strings in the system font (which can vary according to country):

```
#include "SHOW.h"

void main(register ShowRecord *res) {
long margin,height;
Rect dst;

switch(res->message) {
   case shDeclareVersion:
      res->version = 0L;
      break;

   case shDrawResource:
      /* Just use the user's current default text style */
      height = res->fInfo.ascent + res->fInfo.descent +
                                        res->fInfo.leading;
      SetRect(&dst,
            res->leftMargin, 0, res->bounds->right, height);
      /*
       * Center dst vertically in cell, unless there's a
       * name, in which case both must fit.  This SHOW code
       * resource should therefore have a cell height of 24
       * (twice the default height for a line of text).
       */
      if (*res->nameStr)
         margin = 0;
       else
         margin = ((res->bounds->bottom - res->bounds->top)
                                        - height) / 2;
      OffsetRect(&dst, 0, res->bounds->top + margin);
      MoveTo(dst.left, dst.bottom-res->fInfo.descent);
      HLock(res->hndl);
      DrawChar('"'); DrawString(*res->hndl); DrawChar('"');
      HUnlock(res->hndl);

      if (margin == 0)
         MoveTo(dst.left, res->bounds->bottom-res-
>fInfo.descent);
      break;

   } /* End of switch */

} /* End of main */
```

Option-clicking on the resource type, 'STR ', toggles whether to use any available 'SHOW' resource.



If a string resource has a resource name, the 'SHOW' resource lets it be drawn under the value of the string.