Help Manager

Contents

About the Help Manager 3-6
How the Help Manager Displays Balloons 3-8
Default Help Balloons for Menus, Windows, and Icons 3-13
About BalloonWriter 3-17
Using the Help Manager 3-18
Providing Text or Pictures for Help Balloons 3-18
Defining Help Messages 3-19
Using Clear, Concise Phrases 3-20
Using Active Constructions 3-22
Using Parallel Structure 3-22
Offering Hints 3-22
Using Consistent Terminology 3-23
Defining the Help Balloon Position 3-23
Specifying the Format for Help Messages 3-23
Specifying Options in Help Resources 3-25
Providing Help Balloons for Menus 3-27
Specifying Header Information for the 'hmnu' Resource 3-32
Specifying Help for Menu Items Missing From the Resource 3-33
Specifying Help for Menu Titles and for Items Dimmed by System Software 3-36
Specifying Help for Menu Items 3-39
Specifying Help for a Changing Menu Item 3-43
Specifying Resources by Item Name 3-45
Providing Help Balloons for Menus You Disable for Dialog Boxes 3-47
Providing Help Balloons for Items in Dialog Boxes and Alert Boxes 3-51
Specifying Header Information for the 'hdlg' Resource 3-54
Specifying Missing-Item Information 3-54
Specifying Help for Items in an Alert or Dialog Box 3-56
Adding a Help Item to an Item List Resource 3-62
Using a Help Item Versus Using an 'hwin' Resource 3-63

Providing Help Balloons for Window Content 3-63
Providing Help Balloons for Static Windows 3-65
Specifying Header Information for the 'hrct' Resource 3-67
Specifying Help for Rectangles in Windows 3-67
Associating Help Resources With Static Windows 3-68
Specifying Header Information for the 'hwin' Resource 3-69
Specifying 'hdlg' or 'hrct' Resources in the 'hwin' Resource 3-69
Providing Help Balloons for Dynamic Windows 3-74
Overriding Help Balloons for Non-Document Icons 3-84
Specifying Header Information for the 'hfdr' Resource 3-85
Specifying Help for an Icon 3-85
Overriding Other Default Help Balloons 3-87
Specifying Header Information for the 'hovr' Resource 3-88
Overriding Default Help 3-88
Adding Menu Items to the Help Menu 3-90
Writing Your Own Balloon Definition Function 3-93
Help Manager Reference 3-95
Data Structures 3-95
The Help Message Record 3-95
The Help Manager String List Record 3-97
Help Manager Routines 3-97
Determining Balloon Help Status 3-98
Displaying and Removing Help Balloons 3-99
Enabling and Disabling Balloon Help Assistance 3-107
Adding Items to the Help Menu 3-108
Getting and Setting the Font Name and Size 3-110
Setting and Getting Information for Help Resources 3-114
Determining the Size of a Help Balloon 3-119
Getting the Message of a Help Balloon 3-122
Application-Defined Routines 3-128
Resources 3-132
The Menu Help Resource 3-132
The Dialog-Item Help Resource 3-140
The Rectangle Help Resource 3-148
The Window Help Resource 3-154
The Finder Icon Help Resource 3-156
The Default Help Override Resource 3-160
Summary of the Help Manager 3-166
Pascal Summary 3-166
Constants 3-166
Data Types 3-168
Help Manager Routines 3-169
Application-Defined Routines 3-170
C Summary 3-170
Constants 3-170
Data Types 3-173
Help Manager Routines 3-173

Application-Defined Routines 3-175
Assembly-Language Summary 3-176
Data Structures 3-176
Trap Macros 3-176
Result Codes 3-177

Help Manager

This chapter describes how you can use the Help Manager to provide your users with Balloon Help online assistance—information that describes the actions, behaviors, or properties of your application's features. When the user turns on Balloon Help assistance, the Help Manager displays small help balloons as the user moves the cursor over areas such as controls, menus, and rectangular areas in your windows. **Help balloons** are rounded-rectangle windows that contain explanatory information for the user. (With tips pointing at the objects they annotate, help balloons look like the balloons used for dialog in comic strips.) You provide **help messages** in the form of descriptive text or pictures that appear inside help balloons. Your help messages should be short and pertinent to the object over which the cursor is located.

For example, when a user moves the cursor to a menu command, a help balloon should point to that command and explain its purpose. The help balloon remains displayed until the user moves the cursor away.

The user turns on Balloon Help online assistance for all applications by choosing the Show Balloons command from the Help menu. All normally available features of your application are still active when Balloon Help is enabled. The help balloons only provide information; the actions that the user performs by pressing the mouse button still take effect as they normally would.

The Help Manager is available in System 7. Use the `Gestalt` function to determine whether the Help Manager is present.

Read this chapter if you want to provide help balloons for your application, desk accessory, control panel, Chooser extension, or other software that interacts with the user. If you offer an additional help facility for your users, you should give users access to your information through the Help menu. This chapter explains how you can add your own menu items to the Help menu to provide one convenient and consistent place for users to look for help information.

You can provide help balloons for your menus, dialog boxes, alert boxes, and non-document icons by simply adding resources to your resource file. To provide help for the content area of windows, you can use either resources or Help Manager routines. Both methods are described in this chapter.

You typically provide help balloons for your application by creating resources—such as the `'hmnu'` resource, which the Help Manager uses when displaying help balloons for your menu items. In the `'hmnu'` resource, you specify help balloons for menu titles and menu items in their enabled and disabled (that is, dimmed) states. Menus are described in the chapter "Menu Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

To provide help balloons for alert boxes and dialog boxes, you typically create an `'hdlg'` resource that specifies help balloons for the various items identified in the item list (`'DITL'`) resource for the alert box or dialog box. If the items include any controls, such as simple buttons, checkboxes, or complex multipart controls, you specify help according to the control's state—active or inactive (that is, dimmed), and checked or not checked (if applicable). For every item that is not a control, you can provide different help balloons depending on whether the item is enabled or disabled—that is, depending on whether you asked the Dialog Manager to return information regarding events in that item. Dialog boxes and alert boxes are described in the chapter "Dialog Manager" in

Inside Macintosh: Macintosh Toolbox Essentials; controls are described in the chapter "Control Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

Depending on whether your windows are static or whether they contain changing or scrolling information, you use Help Manager resources or Help Manager routines to provide the content areas of your windows with help balloons. To provide help balloons for the static windows of your application without modifying its code, you create a resource of type `'hwin'` and another resource of type `'hrct'` or of type `'hdlg'`. The `'hwin'` resource identifies windows by the titles or the `windowKind` values in their window records. To provide help balloons for portions of windows that change or scroll, you must identify, track, and update those portions within your windows, and then use the Help Manager function `HMShowBalloon` to display help balloons for those portions. Windows are described in the chapter "Window Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

This chapter provides a brief description of how the Help Manager displays help balloons. It provides information on the default help balloons and then discusses how to

- use text or a picture for the help message inside a balloon
- create resources for help balloons for menus, dialog boxes, and alert boxes
- create resources for help balloons for windows
- override the default help balloons provided by system software
- add your own menu items to the Help menu
- write your own balloon definition function

About the Help Manager

You can use the Help Manager to provide help for these interface features of your application:

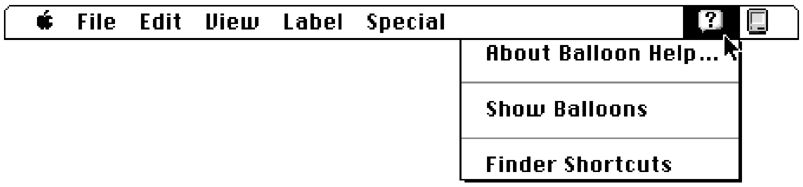
- menu titles and menu items
- dialog boxes and alert boxes
- windows, including any object in the frame or content area
- icons for any desktop objects other than documents
- other application-defined areas

Providing help balloons for menus, dialog boxes, or alert boxes is quite simple, because you need only to create resources; you don't have to alter any of your existing code. The Help Manager automatically sizes, positions, and draws the help balloon and its help message for you. It is equally simple to provide help balloons for a window whose contents don't change location within its content area.

It takes a little more work to provide help balloons for windows in your application that contain objects that are dynamic or that change their position within the content areas of their windows. You provide Balloon Help assistance for these objects by tracking the cursor yourself and using Help Manager routines to display help balloons. You can let the Help Manager remove help balloons, or your application can determine when to remove help balloons.

The user turns on Balloon Help online assistance by choosing Show Balloons from the Help menu, which is shown in Figure 3-1. Once the user chooses Show Balloons, help is enabled for all applications. The Help menu appears to the right of all your menus and to the left of the Application menu (and to the left of the Keyboard menu if a non-Roman script system is installed). Users can turn on Balloon Help assistance even when your application presents an alert box or a modal dialog box, because the Help menu is always enabled.

Figure 3-1 The Help menu for the Finder



When Balloon Help assistance is enabled, the Help Manager displays any help balloons for the current application whenever the user moves the cursor over a rectangular area that has a help balloon associated with it. For those balloons defined in Help Manager resources, the Help Manager automatically tracks the cursor and generates the shape and calculates the position for the help balloon. The Help Manager removes the help balloon when the cursor is no longer located over the associated area.

The Help Manager provides a default help balloon for inactive windows and displays default help balloons for the title bar and other parts of the active window. The Help Manager also displays default help balloons for other standard features of an application's user interface. "Default Help Balloons for Menus, Windows, and Icons" beginning on page 3-13 describes the default help balloons. (Though you probably won't want or need to change the messages in these default balloons, you have the ability to do so, as described in "Overriding Other Default Help Balloons" on page 3-87.) The Help Manager displays the default help balloons for your application whenever Balloon Help assistance is enabled, even if your application does not explicitly use or create help balloons.

Help balloons do not interfere with your application. Because the Help Manager can display a balloon whether the mouse button is down or up, the user can still click and double-click to use the normal features of your application.

When the user chooses Hide Balloons from the Help menu, the Help Manager removes any visible help balloon and stops displaying help balloons until Balloon Help assistance is enabled again.

How the Help Manager Displays Balloons

The Help Manager performs most of the work involved with rendering help balloons for your application. This section gives an overview of the facilities that the Help Manager uses to display help balloons.

The Help Manager uses the Window Manager to create a special type of window for the help balloon and then draws the help message in the port rectangle of the window. The Help Manager is responsible for

- calculating the size of the help balloon (based on the help message you provide)
- determining line breaks for text in a help balloon
- calculating a position for the help balloon so that it appears onscreen
- drawing the help balloon and your help message onscreen

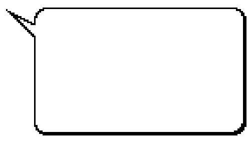
A **balloon definition function,** which is an implementation of a window definition function, defines the general appearance of the help balloon. A standard balloon definition function is provided for you, and it is responsible for

- calculating the help balloon's content region and structure region, which are based on the rectangle calculated by the Help Manager
- drawing the frame of the help balloon

For help balloons, the content region is the area inside the balloon frame; it contains the help message. The structure region is the boundary region of the entire balloon, including the content area and the pointer that extends from one of the help balloon's corners.

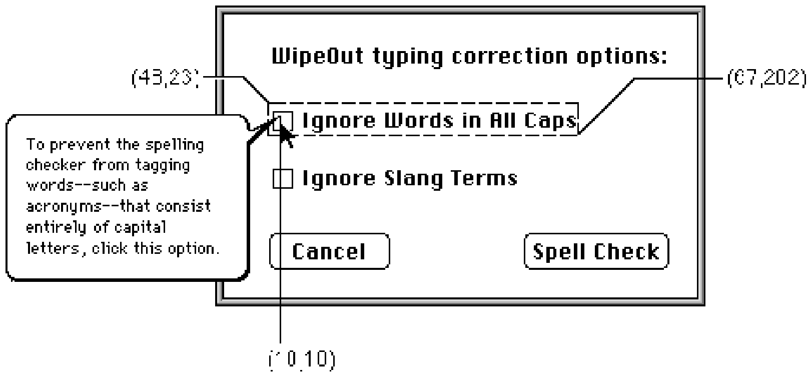
The standard balloon definition function is the window definition function (a `'WDEF'` resource) with resource ID 126. Figure 3-2 shows the general shape of a help balloon drawn with this standard balloon definition function.

Figure 3-2 A help balloon drawn with the standard balloon definition function



Every help balloon is further defined by its hot rectangle, its tip, and a variation code.

Figure 3-3 The tip and hot rectangle for a help balloon



To prevent the spelling checker from tagging words--such as acronyms--that consist entirely of capital letters, click this option.

The **hot rectangle** encloses the area for which you want to provide Balloon Help online assistance. When the user moves the cursor over a hot rectangle, the Help Manager displays the rectangle's help balloon; the Help Manager removes the help balloon when the user moves the cursor away from the hot rectangle. To prevent balloons from flashing excessively, the Help Manager does not display a balloon unless the user leaves the cursor at the same location for a short time (around one-tenth of a second). This length of time is set by the system software and cannot be changed.

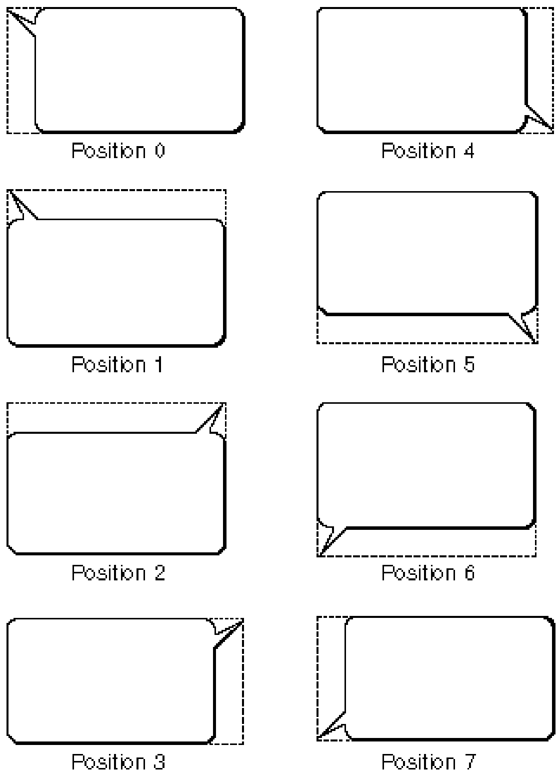
In Figure 3-3, the help balloon is displayed for a hot rectangle defined by coordinates (48,23,67,202), which are local to the window. The Help Manager displays and removes the help balloon as the cursor moves in and out of the area defined by the hot rectangle.

A small pointer extends from a corner of every help balloon, indicating the object or area that is explained in the help balloon. The **tip** is the point at the end of this extension. Figure 3-3 shows an example of a help balloon for a control. The balloon tip is at the coordinates (10,10), which are local to the hot rectangle.

A **variation code** specifies the preferred position of the help balloon relative to the hot rectangle. The balloon definition function draws the frame of the help balloon based on that variation code.

As shown in Figure 3-4, the standard balloon definition function provides eight different positions, which you can specify with a variation code from 0 to 7. The figure also shows the boundary rectangle for each shape. Note that the tip of the help balloon always aligns with an edge of the boundary rectangle. If you write your own balloon definition function, you should support the tip locations defined by the standard variation codes.

Figure 3-4 Standard balloon positions and their variation codes

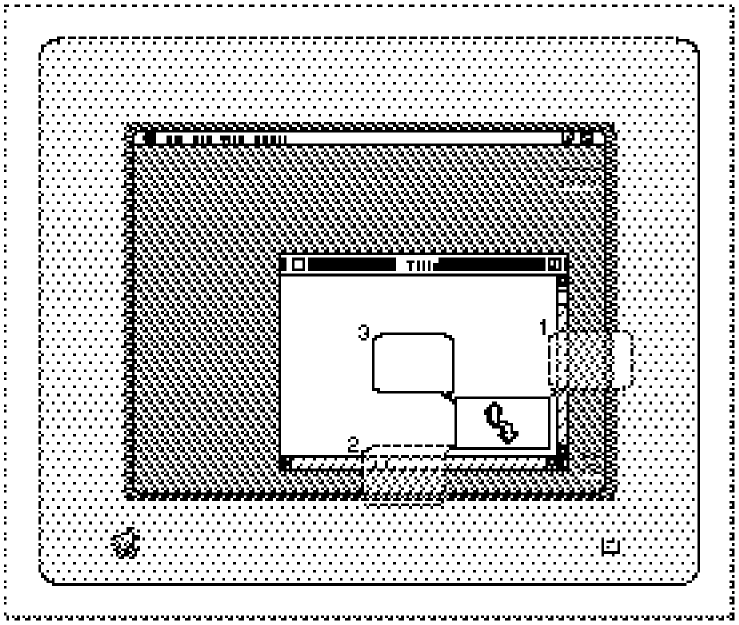


For most of the help balloons it displays, the Finder uses variation code 6. A balloon with variation code 6 has its tip in the lower-left corner and projects up slightly and to the right.

If a help balloon is on the screen and not in the menu bar, the Help Manager uses the specified variation code to display the help balloon. If a help balloon is offscreen or in the menu bar, the Help Manager attempts to display the help balloon by using different variation codes and by moving the tip. Usually, the Help Manager moves the tip by transposing it across the horizontal and vertical planes of the hot rectangle.

Figure 3-5 shows the Help Manager making three attempts to fit a help balloon onscreen by moving the tip to three different sides of the hot rectangle and using an appropriate variation code for each tip.

Figure 3-5 Alternate positions of a help balloon



When positioning a help balloon onscreen, the Help Manager first determines whether the screen has enough horizontal space and then enough vertical space to display the balloon using the specified variation code and tip. If the help balloon is either too wide or too long to fit onscreen at this position, the Help Manager tries a combination of different variation codes and transpositions around the hot rectangle. In Figure 3-5, the Help Manager uses a new variation code, moves the tip to a different side of the hot rectangle, and again tests whether the help balloon fits onscreen. If, after exhausting all possible positions, the Help Manager still cannot fit the entire help balloon onscreen, the Help Manager displays a help balloon at the position that best fits onscreen and clips the help message to fit the balloon at this position.

When you use dialog-item help (`'hdlg'`) resources or the `HMShowBalloon` and `HMShowMenuBalloon` functions, the Help Manager allows you to specify **alternate rectangles,** which give you additional flexibility in positioning your help balloons onscreen. The Help Manager uses the alternate rectangle instead of the hot rectangle for transposing help balloons to make them fit onscreen. If you make your alternate rectangle smaller than your hot rectangle, for example, you have greater assurance that the Help Manager will be able to fit the help balloon onscreen; if you specify an alternate rectangle that is larger than your hot rectangle, you have greater assurance that the help balloon will not obscure some object explained by the balloon.

To provide help balloons under most circumstances, you create **help resources,** which specify the help messages, the balloon definition functions, the variation codes, and, when necessary, the tips and the hot rectangles or alternate rectangles for the Help Manager to use in drawing help balloons. These help resources are

- the menu help (`'hmnu'`) resource, which provides help balloons for menus and menu items
- the dialog-item help (`'hdlg'`) resource, which provides help balloons for items in dialog boxes and alert boxes
- the rectangle help (`'hrct'`) resource, which associates a help balloon with a hot rectangle in a static window
- the window help (`'hwin'`) resource, which associates an `'hrct'` or `'hdlg'` resource with a hot rectangle in a window or with an item in a dialog box or alert box
- the Finder icon help (`'hfdr'`) resource, which provides a custom help balloon message for your application icon
- the default help override (`'hovr'`) resource, which overrides the help messages of default help balloons provided in system software

To put help balloons in your application, you have a number of responsibilities:

- You must create any necessary help resources for your application.
- You must provide the help messages that appear in the balloons. Although you can store these messages in the help resources themselves or in data structures, localizing your help messages is much easier if you store them in other resources—such as `'PICT'`, `'STR#'`, `'STR '`, `'TEXT'`, and `'styl'` resources—that are easier to edit.
- In your help resources you must specify a balloon definition function for your help balloons. Typically, you should use the standard balloon definition function that draws shapes similar to that shown in Figure 3-2 on page 3-8. This helps maintain a consistent look across all help balloons used by the Finder and other applications. However, if you feel absolutely compelled to change the shape of help balloons in your application, you can write your own balloon definition function as described in “Writing Your Own Balloon Definition Function” on page 3-93. Be aware, however, that a different help balloon shape may initially confuse your users.
- In your help resources you must specify a variation code. The variation code positions your balloons onscreen according to the general shape described by their balloon definition function. If you use the standard balloon definition function, you’ll use variation codes 0 to 7 to display the help balloons shown in Figure 3-4 on page 3-10. The preferred variation code is 0. If you are unsure of which variation code to use, specify 0; the Help Manager will use a different variant if another is more appropriate. If you write your own balloon definition function, you must define your own variation codes.

For objects other than menu items, you have these additional responsibilities:

- In your help resources you must specify coordinates for the balloon's tip. For menu items, the Help Manager automatically places the tip just inside the right edge of the menu item.
- You must specify rectangles in your help resources. (The hot rectangles for items in menus, alert boxes, and dialog boxes are automatically defined for you by their display rectangles.) For `'hdlg'` resources, you specify alternate rectangles for moving the help balloon. For `'hrct'` resources, you specify hot rectangles, which define the areas onscreen for association with help balloons.
- You must track the cursor in dynamic windows, and, when the cursor moves over a hot rectangle in your window, you must call Help Manager routines (such as `HMShowBalloon`) to display your help balloons. You can let your application or the Help Manager remove the help balloon when the user moves the cursor out of the hot rectangle.

In summary, the Help Manager automatically displays help balloons in the following manner. The user turns Balloon Help assistance on, then moves the cursor to an area described by a hot rectangle. The Help Manager calculates the size of the help balloon based on its help message. The Help Manager uses TextEdit to determine word breaks and line breaks of text in the help balloon. The Help Manager then determines the size of the help balloon and uses the Window Manager to create a new help balloon. The Window Manager calls the balloon definition function to determine the help balloon's general shape and position. (If the variation code places the help balloon offscreen or in the menu bar, the Help Manager tries a different variation code or moves the tip of the help balloon to another side of the hot rectangle or the alternate rectangle.) The window definition function draws the frame for the help balloon, and the Help Manager draws the help message of the help balloon.

For most interface features that you want to provide help for, you create the help message (preferably in a separate, easily edited resource) and, in the help resources themselves, you specify the standard balloon definition function, one of the eight variation codes, the tip's coordinates, and (often) a hot rectangle.

The Help Manager does not automatically display help balloons for dynamic windows or for menus using custom menu definition procedures. If you want to provide help balloons for either of these types of objects, or if you want more control over help balloons, you must identify hot rectangles, create your own data structures to store their locations, track the cursor yourself, and call `HMShowBalloon` when the cursor moves to your hot rectangles. If you wish, you can also write your own balloon definition function and tip function.

Default Help Balloons for Menus, Windows, and Icons

The Help Manager displays many default help balloons for an application when help is enabled and the user moves the cursor to certain standard areas of the user interface. These areas include the standard window frame and the menu titles and menu items in the Apple menu, Help menu, Keyboard menu, and Application menu. You don't need to create any resources or use any Help Manager routines to take advantage of the default help balloons.

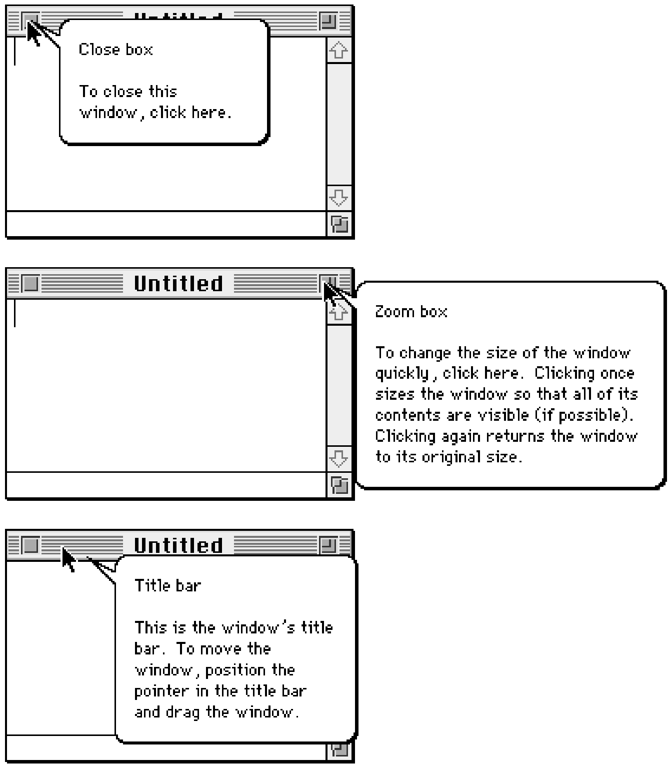
The following list summarizes the items that have default help balloons.

- Application icon in the Finder. Default help balloons are also provided for desk accessory, system extension, and control panel icons. You can override these help messages.
- Document icon in the Finder. You cannot override the help message for this icon.
- Standard file dialog boxes. You supply balloons for items that you add to these dialog boxes; you cannot override the help messages for the other items.
- Window title bar. A default help balloon is provided for the title bars of windows created with both standard and custom window definition functions (WDEFs). You can override the default help message.
- Window close box. A default help balloon is provided for the close boxes of windows created with both standard and customized WDEFs. You can override the default help message.
- Window zoom box. A default help balloon is provided for the zoom boxes of windows created with both standard and customized WDEFs. You can override the default help message.
- Inactive window. You can override the default help message for inactive windows.
- Apple menu title. The default help balloon for the title of the Apple menu is available only if your application uses the standard menu definition procedure. You cannot override the default help message for this title.
- Apple menu items. Default balloons are provided for items that the user moves to the Apple Menu Items folder, but there is no default balloon for the About command or other items that your application adds to this menu; you must provide help balloons for such items.
- Help menu title. The default help balloon for the title of the Help menu is available only if your application uses the standard menu definition procedure. You cannot override the default help message for this title.
- Help menu items. Default balloons are provided only for the About Balloon Help and Hide/Show Balloons commands; you must provide help balloons for items you add to this menu. You cannot override the default help messages.
- Application menu title and items. Default help balloons for the title and items of the Application menu are available only if your application uses the standard menu definition procedure. You cannot override these default help messages.
- Keyboard menu title. The default help balloon for the title of the Keyboard menu is available only if your application uses the standard menu definition procedure. You cannot override the default help message.

System software uses the Help Manager to display help balloons for most of its dialog boxes and alert boxes. (For example, the Standard File Package provides help balloons for its standard file dialog boxes.) If your application uses a system software routine (such as the `StandardPutFile` procedure) that provides help balloons, and the user has enabled Balloon Help assistance, the Help Manager displays each help balloon as the user moves the cursor to each hot rectangle. If you've added your own buttons, checkboxes, or other controls to such a dialog box or alert box, you can also provide these controls with help balloons.

The Help Manager uses the window definition function of a window to determine whether the cursor is in the window frame and, if so, which region of the window (title bar, close box, or zoom box) the cursor is in. If the cursor is in any of these regions, the Help Manager displays the associated help balloon. Figure 3-6 shows the default help balloons for the active window of an application that uses the standard window definition function. If you use a custom window definition function, the Help Manager also displays these default help balloons for the corresponding regions of your windows.

Figure 3-6 Default help balloons for the window frame



Close box

To close this window, click here.

Zoom box

To change the size of the window quickly, click here. Clicking once sizes the window so that all of its contents are visible (if possible). Clicking again returns the window to its original size.

Title bar

This is the window's title bar. To move the window, position the pointer in the title bar and drag the window.

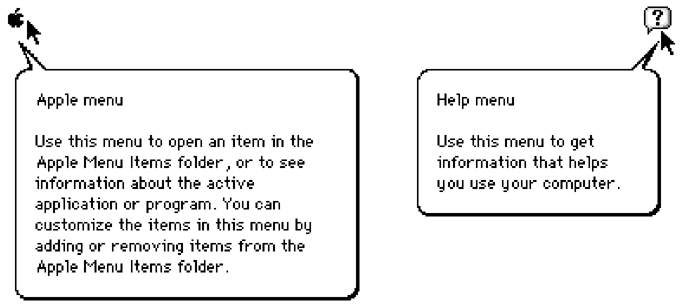
The Help Manager also provides these default help balloons for the title bars, close boxes, and zoom boxes of windows in the Finder. The Finder specifies additional help for other window regions—for example, the scroll bar and size box—although the Help Manager does not automatically provide your window with this help.

The Help Manager displays help balloons for the standard window frame and other standard areas named in the `'hovr'` resource. You can override any of the default help balloons defined in the `'hovr'` resource by providing your own resource of type `'hovr'`. See "Overriding Other Default Help Balloons" on page 3-87 for more information.

The Help Manager displays default help balloons for the Apple menu, Help menu, and Application menu. The Menu Manager uses the Help Manager to display help balloons for these menus regardless of whether you supply help balloons for the rest of your menus. The Help Manager also provides default help balloons for the Keyboard menu when a non-Roman script system is installed. Figure 3-7 shows the default help balloons for the Apple menu and Help menu titles.

Note
For all menus and menu items, the Help Manager displays help balloons only for applications that use the standard menu definition procedure. If you use your own menu definition procedure, your application must track the cursor and use Help Manager routines to display and remove help balloons, as described in "Displaying and Removing Help Balloons" on page 3-99. ◆

Figure 3-7 Default help balloons for the Apple and Help menus


Apple menu

Use this menu to open an item in the Apple Menu Items folder, or to see information about the active application or program. You can customize the items in this menu by adding or removing items from the Apple Menu Items folder.

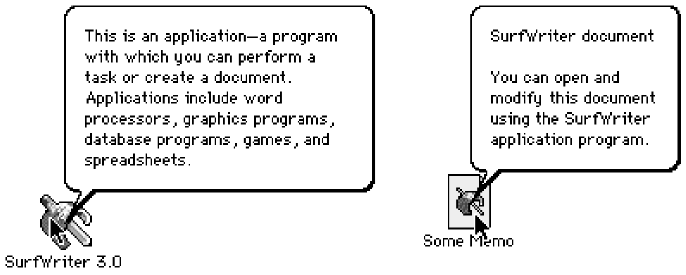
Help menu

Use this menu to get information that helps you use your computer.

The Help Manager does not provide default help balloons for items you put at the top of your application's Apple menu or items you add to the Help menu. You typically put one item at the top of the Apple menu: the About command for your application. If you have additional user help facilities, list them in the Help menu—not in the Apple menu. You have control only over those items that you add to the Apple and Help menus.

The Finder provides default help balloons for your application icon and any documents created by your application. Figure 3-8 shows the default help balloon for the SurfWriter application and a document created by this application. You can customize the help balloon for your application icon by providing an `'hfdr'` resource; however, you can't customize the default help balloon for the documents created by your application.

Figure 3-8 Default help balloons for application and document icons


This is an application—a program with which you can perform a task or create a document. Applications include word processors, graphics programs, database programs, games, and spreadsheets.
SurfWriter 3.0
SurfWriter document

You can open and modify this document using the SurfWriter application program.
Some Memo

About BalloonWriter

Apple Computer, Inc., makes available a tool that greatly facilitates the creation of help balloons. Called BalloonWriter, this tool gives nonprogrammers an easy, intuitive way to create help balloons. Writers who have no programming experience can use BalloonWriter to provide your application with fully functional resource code for menus, dialog and alert boxes, static windows, and non-document Finder icons. In its user's guide, BalloonWriter refers to help balloons for these interface features as *standard balloons*. For these types of help balloons, BalloonWriter creates `'hmnu'`, `'hdlg'`, `'hwin'`, `'hrct'`, and `'hfdr'` resources, as appropriate, and places them in the resource file of your application. BalloonWriter likewise creates and stores `'STR '`, `'STR#'`, and `'TEXT'` resources that contain the help messages authored by your nonprogramming writers.

For dynamic windows and for menus that use custom menu definition procedures, your application must track the cursor and use the `HMShowBalloon` function to display help balloons. The BalloonWriter documentation refers to these balloons as *custom balloons*. BalloonWriter does not create the necessary resources or code that automatically displays these types of help balloons. However, nonprogrammers can use BalloonWriter to provide you with conveniently delimited ASCII text that you can then use in conjunction with `HMShowBalloon` to display the desired help balloons.

BalloonWriter is available from APDA.

Using the Help Manager

You can use the Help Manager to provide information to the user that describes the action, behavior, or properties of your application's features. For example, you can create a help balloon for each menu item to describe what it does.

To determine whether the Help Manager is available, use the `Gestalt` function with the `gestaltHelpMgrAttr` selector. Test the `bit` field indicated by the `gestaltHelpMgrPresent` constant in the `response` parameter. If the bit is set, then the Help Manager is present.

```
CONST gestaltHelpMgrPresent   = 0;     {if this bit is set, then }
                                       { Help Manager is present}
```

The Help Manager is initialized at startup time. The user controls whether help is enabled by choosing the Show Balloons or Hide Balloons command from the Help menu.

The Help menu is specific to each application, just as the File and Edit menus are specific to each application. The Help menu items that are defined by the Help Manager are common to all applications, but you can add your own menu items for help-related information.

The Help Manager automatically appends the Help menu when your application inserts an Apple menu into its menu bar. The Menu Manager automatically appends the Help menu to the right of all your menus and to the left of the Application menu (and to the left of the Keyboard menu if a non-Roman script system is installed).

You can create help balloons for the menus, dialog boxes, alert boxes, or content area of windows belonging to your application. You can also override some of the default help balloons—such as the default help balloon for the title bar of a window.

You can specify the help message by using plain text, styled text, or pictures. Although you should always strive for brevity in your help messages, plain text strings can contain up to 255 characters. Styled text can contain up to 32 KB of information. The Help Manager determines the actual size of the help balloon and, for text strings, uses TextEdit to determine word breaks and line breaks.

The Help Manager automatically tracks the cursor and generates help balloons defined in standard help resources. Your application can also track the cursor and use Help Manager routines to display and remove help balloons.

Providing Text or Pictures for Help Balloons

Use help balloons to provide the user with information that describes or explains interface features of your application. The information you supply in help balloons should follow a few general guidelines in order to provide the most useful information to the user. This section describes these guidelines.

For examples of how your application should use help balloons, observe the help balloons that the Finder, the TeachText application, and system software use.

Defining Help Messages

Use help balloons to explain parts of your application's interface that might confuse a new user or features that could help a user become an expert user. The information you provide in help balloons should identify interface features in your application or describe how to use them.

The help balloon for an item appears when the user moves the cursor to that item. Because the user knows exactly what the text is referring to, this is a powerful method of providing information. But the method has some limitations. There are some kinds of information that help balloons cannot display effectively.

- Help balloons *can* show users what they will accomplish by using onscreen objects, including menu commands, dialog boxes, and tool palettes.
- Help balloons *can* help experienced Macintosh users who prefer to learn programs by using them, rather than by reading manuals.
- Help balloons *can't* help users who don't know what they want to do or users who don't know where to look.
- Help balloons *can't* teach your program by themselves. They can't substitute for task-oriented paper or electronic documentation or training.
- Help balloons *can't* teach novice Macintosh users the concepts they need to know in order to use the Macintosh computer.

Help balloons work best when you keep your audience in mind as you write. Ask yourself these questions when you are planning balloons for your program:

- Who will be using your program?
- What aspects of your program are users unfamiliar with?
- What terminology are your users likely to know?

Unless your application has a specialized audience, it's best to write for users who already know something about using the Macintosh (although they may not be experts) but who don't know much about your application.

Each help balloon should answer at least one of these questions:

- **What is this?** For example, when the user moves the cursor to the item count in the upper-right corner of a Finder window, the Finder displays a help balloon that reads "This is the number of files or folders in this window."
- **What does this do?** For example, when the user moves the cursor to the Find command in the Finder's File menu, the Finder displays a help balloon that reads "Finds and selects items with the characteristics you specify."
- **What happens when I click this?** For example, when the user moves the cursor to the close box of a window, the Window Manager displays a help balloon that first names the object ("Close box") and then explains, "To close this window, click here."

Help messages should be short and easy to understand. You should not include lengthy instructions or numbered steps in help balloons. Use help balloons to clarify the meaning of objects in your application—for example, tool icons in palettes.

Use simple, clear language in the information you provide. Include definitions in help balloons when appropriate.

You can use graphics or styled text in help balloons to illustrate the effects of a command. For example, to demonstrate the effect of the Bold command in a word-processing application, you might use styled text to show a word in boldface.

You can provide separate help balloons for two display states—enabled and dimmed (disabled)—of a menu item. You can also provide separate help balloons for two display states—active and dimmed (inactive)—of a control. The help balloon that you provide for an enabled menu item should explain the effect of choosing the item. The help balloon that you provide for a dimmed menu item should explain why it isn't currently available, or, if more appropriate, how to make it available. Similarly, the help balloon that you provide for an active control should explain the effect of clicking or selecting the control, and the help balloon that you provide for a dimmed control should explain why it isn't currently available, or, if more appropriate, how to make it available.

Complicated dialog boxes can often benefit from help balloons that explain what's essential about the dialog box. You can use help balloons to describe groups of controls rather than individual controls. For example, if a dialog box has several distinct regions that contain radio buttons or checkboxes, you could provide a help balloon for each set of radio buttons, rather than providing a separate balloon for each button.

If you use a function to customize standard dialog boxes, use as many of the existing help balloons as possible. For example, if your application uses any of the standard file dialog boxes and provides an extra button, you can create a help balloon for the extra button, and the Help Manager continues to use the default help balloons for other items in the dialog box.

To make localization easier, you should store your help messages in resources separate from the help resources. To avoid problems with grammar and sentence structure when you localize your application, never combine separately stored phrases into one help message.

Using Clear, Concise Phrases

You can provide up to 255 characters of information using text strings in help balloons. (You can use up to 32 KB if you use styled text.) However, you should include only the most relevant information in the help balloon. To determine what to provide, decide what information would be most useful to a user. This information usually omits the object's name, which normally doesn't matter to the user, and instead tells what the object is for and what the object does, which does matter to the user.

You might eventually translate your help messages into other languages, so try to keep the messages as short as possible. When translated, your help messages may require more words or longer words—and therefore larger balloons and more screen space. Expect English text to expand 20–30 percent after translation. To keep the translated text within the Help Manager's 255-character limit for text strings, limit English text to approximately 180 characters.

If an item already has a commonly used name, or if it's a special case of a larger category of objects, name it in the balloon. The Finder, for example, displays the message "Drag the title bar to move the window," since title bars and windows are commonly used names. However, you don't need to name everything in your application just so that you can refer to it in a help balloon. For example, because the tip of the help balloon points to the subject of the help balloon, you can easily say "To apply the style, click here," rather than "The Apply button activates the Styles command. Click the button to activate the command."

Many of the items onscreen don't need names. An item needs a name only if the name helps the user remember how to use the application. The following items are likely to need names:

- icons that don't already have names on the screen
- tools in a palette
- controls on a ruler
- controls in a paint program
- Finder icons whose names can be changed

If you decide to name an item, make sure that the name you use in the balloon matches the name used in other documentation.

For balloons that describe menu items, you can use sentence fragments; the grammatical subject is obvious from the context. For example, the help balloon for the Open command could read "Opens the selected file" rather than "This command opens the selected file"; the grammatical subject is obvious from the context. Using sentence fragments lets users assimilate the message more quickly because they have fewer words to read.

When you describe a menu item or a button, try to use a word that's different from the one that appears onscreen. Using a synonym in this way helps users who aren't sure what the item's name means. For example, the help balloon for a Paste command in the Edit menu might say something like "Inserts the contents of the Clipboard into the document."

Help balloons are usually inappropriate for describing multiple-step procedures, because a help balloon does not stay on the screen while the user performs the various steps. The user may begin a procedure described in a help balloon and then become confused when the information disappears.

You can, however, describe a very simple two-step procedure in a balloon. This is probably most appropriate for a tool in a palette. For example, the balloon for an eraser tool might first define the tool as an eraser and then explain, "To remove parts of your drawing, click this icon, then drag to erase those parts you want to remove."

Using Active Constructions

Try to use short, active phrases in help balloons. Avoid passive constructions. An active construction is more forceful because it communicates how the grammatical subject (usually the user in this context) performs an action. In the sentence "To turn the page, click here," the implied "you" (that is, the user) is the subject, and "click" is the action that the subject performs. Passive constructions show subjects being acted upon rather than performing an action. For example, in the sentence "The page will be turned when this button is clicked," both "page" and "button" are acted upon.

Research suggests that instructional materials are more effective when they present the goal clause before the action clause, helping readers quickly recognize how the information meets their needs. A goal might be "To turn the page," "To calculate the result," or "To apply the style." For example, the message "To turn the page, click here" starts with a goal statement and then describes the action necessary to fulfill it; users find this more helpful than a purely descriptive message like "This button turns the page."

If there is more than one way for the user to achieve a goal, mention only the method that involves the item to which the user is pointing. In other words, if the user is pointing to a button, the balloon should tell the user how to use the button, not how to use a keyboard shortcut for that button. For example, a help balloon for a Save button might state, "To save the changes you have made to the settings in the dialog box, click this button"—but the help balloon should *not* add "or press the Return key."

If there is more than one method for using the item to which the user is pointing, describe the method that's simplest to explain and understand.

Using Parallel Structure

Use similar syntax for help balloons that describe similar objects. For example, all help balloons that describe buttons should have the same structure. In a style dialog box, you might provide these messages for the buttons: "To see the style, click Apply," "To implement the style, click OK," and "To do nothing to change the previous style, click Cancel."

Users see help balloons provided by many different applications, so a consistent approach within your application helps them to identify types of balloons quickly and to develop realistic expectations about their help messages.

Offering Hints

If there are just a few interesting features in your application that would be difficult to discover, then it's appropriate to use balloons to call those features to users' attention.

But if you want to give a hint or shortcut in a balloon, ask yourself these questions:

- Is the balloon reasonably short, even with the hint?
- How often will users need the information? If a feature is very obscure and few people will need it, the balloon probably shouldn't describe it.

- Are hints and shortcuts available somewhere else—for example, in a "shortcuts" dialog box or a quick-reference card? Not all users will look at balloons. If your program includes many shortcuts and tricks, be sure to list them in other documentation as well.
- Does the need for hints indicate the need for a different design? If your application contains many hidden shortcuts and features, then you may need to redesign your application to make these features more easily accessible to users.

If you include a hint or shortcut, put the hint at the bottom of the balloon and separate it from the rest of the message by a blank line. For example, the Clean Up Window command in the Finder's Special menu initially describes the command's effect: "Neatly arranges the icons in the active window." Then there is a blank line followed by a hint: "Tip: for other cleanup commands, hold down the Shift or Option key while choosing this command."

Using Consistent Terminology

You should employ consistent terminology in all your help balloons. Use language that users understand; avoid introducing technical jargon or computer terminology into help balloons. Follow the style and usage standardized by Apple Computer, Inc., in the *Apple Publications Style Guide* (available through APDA) to make the most effective use of the information and vocabulary with which users are already familiar. A supplement to the *Apple Publications Style Guide*, titled "How to Write Balloons," spells out the guidelines that Apple writers use for the wording and phrasing of help messages. This supplement also provides many examples of clear and useful help messages as well as counterexamples of types of messages to avoid.

Defining the Help Balloon Position

When you provide a help balloon, you specify its help message, the tip of the help balloon, and the variation code for its preferred position. The tip of the help balloon should point to the object that the help balloon describes. You should specify the tip and the variation code so that the help balloon doesn't obscure the object for which you're providing help. In most cases, the tip of the help balloon should point to an edge of the object.

You should also consider how the Help Manager repositions the balloon if the variation code places it offscreen. "How the Help Manager Displays Balloons" on page 3-8 describes how the Help Manager repositions the help balloon if necessary.

Specifying the Format for Help Messages

You specify the format for your help messages as text strings within help resources, as text strings within `'STR '` resources, as lists of text strings within `'STR#'` resources, as styled text using `'TEXT'` and `'styl'` resources, or as pictures described in `'PICT'` resources.

Later sections in this chapter describe all the help resources in detail. Common to all the help resources are the following identifiers, by which you identify the format of your help messages.

Identifier	Help message format
`HMStringItem`	A text string (a Pascal string stored in the help resource)
`HMPictItem`	A picture (stored in a `'PICT'` resource)
`HMStringResItem`	A text string (stored in a list of strings as an `'STR#'` resource)
`HMTEResItem`	Styled text (stored in both a `'TEXT'` and an `'styl'` resource)
`HMSTRResItem`	A text string (stored in an `'STR '` resource)
`HMSkipItem`	No help message—skip this item

You specify the identifiers within the help resources; the Help Manager reads these identifiers to determine where and how your help messages are stored. You can use the `HMStringItem` identifier to store Pascal strings directly in a help resource. However, you can make it much easier to localize your product by storing your help messages in separate resources—namely, in `'STR#'`, `'PICT'`, `'STR '`, and `'TEXT'` resources—that can be modified by nonprogrammers using tools like BalloonWriter and the ResEdit resource editor.

To display a diagram or illustration in `'PICT'` format, use the `HMPictItem` identifier. You provide a help message by specifying the resource ID of the `'PICT'` resource that contains the diagram or illustration, and the Help Manager displays the picture in a help balloon.

To display a text string stored in a string list (`'STR#'`) resource, use the `HMStringResItem` identifier. You provide a help message by specifying two items in your help resource: the resource ID of an `'STR#'` resource, and the index to a particular text string from within that list. For more information on these items, see “Providing Help Balloons for Menus” beginning on page 3-27.

To display styled text, use the `HMTEResItem` identifier. You provide a help message by specifying a resource ID that is common to both a style scrap (`'styl'`) resource and a `'TEXT'` resource, and the Help Manager employs TextEdit routines to display your text with your prescribed styles. For example, you might create a `'TEXT'` resource with resource ID 1000 that contains the words “Displays your text in boldface print” and a `'styl'` resource with resource ID 1000 that applies boldface style to the message. (See the chapter “TextEdit” in *Inside Macintosh: Text* for a description of the style scrap.)

To display text from a simple text string (`'STR '`) resource, use the `HMSTRResItem` identifier. You provide a help message by specifying the resource ID of an `'STR '` resource, and the Help Manager displays the text from that resource in a help balloon. With `'STR '` resources, each text string must be stored in a separate resource. It is usually more convenient to group related help messages in a single `'STR#'` resource and use the `HMStringResItem` identifier as previously described.

You can use the `HMSkipItem` identifier for items for which you don’t want to provide a help balloon. For example, you specify `HMSkipItem` for the divider lines that appear in menus. (Divider lines cannot have help balloons.)

Specifying Options in Help Resources

Each help resource contains an element that allows you to specify certain options. Notice the options element in the following header component for an 'hmnu' resource.

```
resource 'hmnu' (130, "Edit", purgeable) {
   HelpMgrVersion,      /*version of Help Manager*/
   hmDefaultOptions,    /*options*/
   0,                   /*balloon definition function*/
   0,                   /*variation code*/
```

You should normally use the hmDefaultOptions constant, as shown in the preceding example, to get the standard behavior for help balloons. However, you can also use the constants listed here for the options element. (Note that not all options are available for every help resource.)

```
CONST hmDefaultOptions     = 0;  {use defaults}
      hmUseSubID           = 1;  {use subrange resource IDs }
                                 { for owned resources}
      hmAbsoluteCoords     = 2;  {ignore coords of window }
                                 { origin and treat upper-left }
                                 { corner of window as 0,0}
      hmSaveBitsNoWindow   = 4;  {don't create window; save }
                                 { bits; no update event}
      hmSaveBitsWindow     = 8;  {save bits behind window }
                                 { and generate update event}
      hmMatchInTitle       = 16; {match window by string }
                                 { anywhere in title string}
```

If you're providing help balloons for a desk accessory or a driver that uses owned resources, use the hmUseSubID constant in the options element. Otherwise, the Help Manager treats the resource IDs specified in the rest of your help resource as standard resource IDs. (See the chapter "Resource Manager" in this book for a discussion of owned resources and their resource IDs.)

As described later in this chapter, you often specify tip and rectangle coordinates in your help resources. When specifying these coordinates within a scrolling window or whenever the window origin is offset from the origin of the port rectangle, you may want to use the hmAbsoluteCoords constant. This causes the Help Manager to ignore the local coordinates of the port rectangle when tracking the cursor and instead to track the mouse location relative to the window origin. When you specify the hmAbsoluteCoords constant as an option in a help resource, the Help Manager subtracts the coordinates of the window origin from the coordinates of the mouse location and uses the results for the current mouse location, as shown here:

```
mousepoint.h := mousepoint.h – portRect.left;
mousepoint.v := mousepoint.v – portRect.top;
```

With the `hmAbsoluteCoords` constant specified, the Help Manager always assigns coordinates (0,0) to the point in the upper-left corner of the window. So, for example, if the cursor is positioned at point (4,5) in a port rectangle and the window origin is at (3,4), the Help Manager calculates the cursor at (1,1). If this option is not specified, the Help Manager uses the port rectangle's local coordinates when tracking the cursor—for example, when using the `GetMouse` procedure.

The Help Manager draws and removes help balloons in three different ways. For all help resources except `'hmnu'` resources, the Help Manager by default draws and removes help balloons as if they were windows. That is, when drawing a balloon, the Help Manager does not save bits behind the balloon, and, when removing the balloon, the Help Manager generates an update event. By specifying the `hmDefaultOptions` constant in your help resources, you always get the standard behavior of help balloons. However, you can often specify two options that change the way balloons are drawn and removed from the screen.

If you specify the `hmSaveBitsNoWindow` constant for the options element, the Help Manager does not create a window for displaying the balloon. Instead, the Help Manager creates a help balloon that is more like a menu than a window. The Help Manager saves the bits behind the balloon when it creates the balloon. When it removes the balloon, the Help Manager restores the bits without generating an update event. You should use this option only in a modal environment where the bits behind the balloon cannot change from the time the balloon is drawn to the time it is removed. For example, you might choose the `hmSaveBitsNoWindow` option in a modal environment when providing help balloons that overlay complex graphics, which might take a long time to redraw with an update event. Note that the Help Manager always uses this behavior when drawing and removing help balloons specified in your `'hmnu'` resources. That is, when you specify the `hmDefaultOptions` constant in an `'hmnu'` resource, the Help Manager provides this sort of balloon instead of drawing a window for a balloon. (In an `'hmnu'` resource, you cannot even specify options for drawing a window for a balloon.)

If you specify the `hmSaveBitsWindow` constant, the Help Manager treats the help balloon as a hybrid having properties of both a menu and a window. That is, the Help Manager saves the bits behind the balloon when it creates the balloon and, when it removes the balloon, it both restores the bits and generates an update event. You'll rarely need this option. It is necessary only in a modal environment that might immediately change to a nonmodal environment—that is, where the bits behind the help balloon are static when the balloon is drawn, but can possibly change before the help balloon is removed. For example, if you use an `'hmnu'` resource to provide help balloons for menu titles and menu items, you'll notice that the Help Manager automatically provides this sort of behavior (even when you don't specify the `hmSaveBitsWindow` option) when creating help balloons for menu titles.

In the preceding list of constants, the values for the constants represent bit positions that are set to 1. To override more than one default, add the values of the bit positions for the desired options and specify this sum, instead of a constant, for the options element. For example, to use subrange IDs, ignore the window port origin coordinates, and save bits behind the help balloon without generating an update event, you should add the values of the bit positions of these options (1, 2, and 4) and specify their sum (7) for the options element.

If you supply the `hmDefaultOptions` constant, the Help Manager treats the resource IDs in this resource as regular resource IDs and not as subrange IDs; it uses the port rectangle's local coordinates when tracking the cursor; and it generally creates windows when drawing balloons and then generates update events without saving or restoring bits when removing balloons.

The `hmMatchInTitle` constant is used only in window help (`'hwin'`) resources to match windows containing a specified number of characters in their titles. This constant is explained in more detail in "Providing Help Balloons for Static Windows" on page 3-65.

The next sections describe how to create help resources that provide help balloons for the standard user interface features of your application.

Providing Help Balloons for Menus

If your application uses the standard menu definition procedure, you'll find that it's easier to provide help balloons for menus than for any of your other interface features. This section is relatively lengthy compared to the sections describing dialog boxes, alert boxes, and windows because it explains in detail much of the work you'll also perform when supplying help balloons for those items.

This section assumes that your application uses the standard menu definition procedure. If your application uses its own menu definition procedure, you must use Help Manager routines to display and remove help balloons. These routines are described in "Displaying and Removing Help Balloons" on page 3-99. Even if you use these routines, you should read this section so that your balloons emulate the behavior that the Help Manager provides for menus using the standard menu definition function.

To create help balloons for a menu—pull-down, pop-up, or hierarchical—that uses the standard menu definition procedure, create a resource of type `'hmnu'` in which you specify help balloons for the menu title and for each menu item. You create a separate `'hmnu'` resource for each menu.

Note

BalloonWriter, available from APDA, is a tool that gives nonprogrammers an easy, intuitive way to create help balloons for menus. BalloonWriter creates `'hmnu'` resources as appropriate and places them in the resource file of your application; BalloonWriter likewise creates and stores `'STR '`, `'STR#'`, and `'TEXT'` resources that contain the help messages authored by nonprogramming writers. For menus that use custom menu definition procedures, nonprogrammers can use BalloonWriter to provide you with delimited ASCII text that you can then use in conjunction with `HMShowBalloon` to display the desired help balloons. ◆

The Help Manager can display different help balloons for the various states of a menu item. Each menu item can have up to four help balloons associated with it, one for each state:

- enabled
- disabled (that is, dimmed)
- enabled and checked
- enabled and marked (that is, marked by a symbol other than a checkmark—for example, a bullet or a diamond)

For example, you can define a help balloon that the Help Manager displays when the Cut command is enabled and another help balloon for display when the Cut command is dimmed. Remember that the help balloon you provide for a dimmed menu item should explain why it isn't currently available or, if more appropriate, how to make it available.

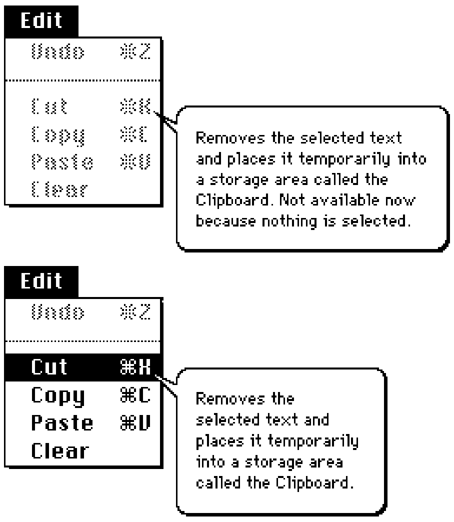
Note

Although `enabled` and `disabled` are the constants you use in a resource file to display or dim menus and menu items, you shouldn't use these terms in your help balloons or user guides. Rather, use the term *menus*, *menu commands*, or *menu items* for those that are enabled, and use the term *not available* or *dimmed* to distinguish those that have been disabled. ◆

When your application calls the Menu Manager function `MenuSelect` or `MenuKey`, the Menu Manager automatically tracks the cursor, highlights enabled menu items, and displays any additional hierarchical or pop-up menus as the user moves the mouse. As the user drags the cursor across or through a menu, the Menu Manager uses the Help Manager to display any help balloons associated with the current state of the menu title or menu item.

If there is sufficient memory, the standard menu definition procedure saves the bits behind the help balloon and restores these bits for quick updating of the screen. If there isn't sufficient memory to save the bits behind the help balloon, then—as with menus—the procedure generates appropriate update events. Figure 3-9 shows help balloons for two instances of a menu, one with the Cut command dimmed, the other with the Cut command enabled.

Figure 3-9 Help balloons for different states of the Cut command


Edit
Undo ⌘Z
Cut ⌘X
Copy ⌘C
Paste ⌘V
Clear
Removes the selected text and places it temporarily into a storage area called the Clipboard. Not available now because nothing is selected.
Edit
Undo ⌘Z
Cut ⌘X
Copy ⌘C
Paste ⌘V
Clear
Removes the selected text and places it temporarily into a storage area called the Clipboard.

You don't specify hot rectangles or tip coordinates for menus. The rectangles defined by the Menu Manager for menu titles and menu items are used as hot rectangles. The Help Manager initially tries to draw a help balloon for a menu item using variation code 0 (shown in Figure 3-4 on page 3-10) with the tip placed 8 pixels inside the right edge and halfway between the top and bottom edges of the menu item's rectangle. If the balloon's initial position lies wholly or partially offscreen, the Help Manager tries to redraw the balloon by moving its tip to the left edge of the item's rectangle and using variation code 3. The Help Manager uses variation codes 1 and 2 in its attempts to draw help balloons for menu titles. The Help Manager never moves the tip for menu titles; instead, the tip is always located just below the bottom of the menu bar at the midpoint of the menu title's text.

The resource ID of each `'hmnu'` resource should match the corresponding menu ID. For example, to provide help balloons for a menu with ID 130, create an `'hmnu'` resource with resource ID 130.

The `'hmnu'` resource contains four types of components, listed below. Each component consists of several elements that contain information used by the Help Manager.

- The **header component** is where you specify information that applies to all help balloons specified in this resource—information such as the version number of the Help Manager, the balloon definition function, and the variation code.
- The **missing-items component** is where you specify help messages for any menu items missing from or unspecified in the rest of the resource. This is useful, for example, for allowing several menu items to share the same help message.

- The **menu-title component** is where you specify help messages for the menu title.
- A **menu-item component** is where you specify the help messages for a particular menu item. You can include any number of menu-item components; however, the menu-item components in the `'hmnu'` resource must appear in the order in which their corresponding menu items appear in the menu. If you do not provide menu-item components for any items at the bottom of a menu, a help message from the missing-items component is used.

Here is the general Rez input format of an `'hmnu'` resource. (Rez is the resource compiler provided with Apple's Macintosh Programming Workshop [MPW], available from APDA.)

Component	**Element**
Header	Help Manager version
	Options
	Balloon definition function
	Variation code
Missing item	Identifier
	Help message for missing enabled items
	Help message for missing items dimmed by application
	Help message for missing enabled-and-checked items
	Help message for missing enabled-and-marked items
Menu title	Identifier
	Help message for enabled menu title
	Help message for menu title dimmed by application
	Help message for menu title dimmed by system software
	Help message for menu items dimmed by system software
First menu item	Identifier
	Help message for enabled menu item
	Help message for menu item dimmed by your application
	Help message for enabled-and-checked menu item
	Help message for enabled-and-marked menu item
Next menu item	(Same as for first menu item)
	.
	.
	.
Last menu item	(Same as for first menu item)

Listing 3-1 shows Rez input code for the `'hmnu'` resource for an Edit menu.

Listing 3-1 Rez input for a partial 'hmnu' resource

```
resource 'hmnu' (130, "Edit", purgeable) {
   /*header component*/
   HelpMgrVersion,
   hmDefaultOptions,       /*options*/
   0,                      /*balloon definition function*/
   0,                      /*variation code*/
   /*missing-items component*/
   HMSkipItem {
      /*no missing items, so skip to menu-title component*/
      },
      { /*menu-title component*/
         HMStringItem { /*use following P-strings*/
            /*use string below when menu is enabled*/
            "Edit menu\n\nUse this menu to manipulate text.",
            /*use string below when app dims menu*/
            "Edit menu\n\nUse this menu to manipulate text. "
               "Not available because you do not have permission "
               "to alter this file.",
            /*use string below for title dimmed by system */
            /* software for an alert or modal dialog box*/
            "Edit menu\n\nUse this menu to manipulate text. "
               "Not available because a dialog box is on "
               "the screen.",
            /*use string below for all items when system */
            /* software dims them for an alert or modal */
            /* dialog box*/
            "This item is not available because a dialog box "
               "is on the screen.",
         },

      /*first menu-item component: Undo command*/
         HMStringItem {/*use following P-strings*/
            /*use string below when command is enabled*/
            "Cancels your last edit.",
            /*use string below when app dims the command*/
            "Cancels your last edit. Not available because "
               "you haven't performed an editing action yet.",
            /*can't check the item, so empty string goes below*/
            "",
            /*can't mark the item, so empty string goes below*/
            "",
         },
      /*second menu-item component: divider line*/
         HMSkipItem { /*no help balloons for divider lines*/
         },
      /*third menu-item component: Cut command*/
         HMStringItem { /*use following P-strings*/
            /*use string below when command is enabled*/
            "Cuts the selected text to the Clipboard.",
            /*use string below when app dims the command*/
            "Cuts the selected text to the Clipboard. "
               "Not available now because no text is selected.",
            /*can't check item, so empty string goes below*/
            "",
            /*can't mark item, so empty string goes below*/
            "",
         }
      /*menu-item components for Copy, Paste, and Clear */
      /* commands go here*/
      }
};
```

Specifying Header Information for the 'hmnu' Resource

The header component of an `'hmnu'` resource consists of these elements:

1. Help Manager version.
2. Options.
3. Balloon definition function.
4. Variation code.

Always specify the `HelpMgrVersion` constant for the Help Manager version element.

For the options element, you must specify the constant `hmDefaultOptions`.

The third element in the header component specifies the resource ID of the window definition function that is used to draw the frame of the help balloon. To use the standard balloon definition function, specify 0 for this element; this is the suggested default. If you use your own balloon definition function (as described in “Writing Your Own Balloon Definition Function” on page 3-93), specify its resource ID for this element.

The fourth element in the header component specifies the preferred position of the help balloon. For example, the standard balloon definition function displays help balloons according to eight different positions. If you specified the standard balloon definition for the preceding element, supply a variation code from 0 to 7 to display the balloon according to one of the eight positions shown in Figure 3-4 on page 3-10. The preferred variation code is 0. If you are unsure of which variation code to use, specify 0; the Help Manager will use a different variant if another is more appropriate. If you use your own balloon definition function, you specify its variation code for this element of the header component.

Specifying Help for Menu Items Missing From the Resource

After the header component, you specify the format and help messages for help balloons for missing items, for the menu title, and for the menu items.

Use the missing-items component of the `'hmnu'` resource to specify how the Help Manager handles menu items that are not described in this resource. You can also use the missing-items component to supply help messages for menu items that are described in the `'hmnu'` resource but that lack help messages for any particular states.

The missing-items component of this resource is useful when you have menu items with similar characteristics or when the number of menu items is variable. For example, if the help message for a dimmed item applies to all dimmed menu items, you can specify a help message once in the third element of the missing-items component instead of repeating it in every third element of the various menu-item components.

The missing-items component consists of the following five elements:

1. An identifier (either `HMStringItem`, `HMSTRResItem`, `HMStringResItem`, `HMPictItem`, `HMTEResItem`, or `HMSkipItem`) for the format of the help messages.
2. The help message when a menu item is enabled. This message is displayed either when the item itself is not specified in a menu-item component of this `'hmnu'` resource or when its help message *is* specified in a menu-item component, but specified with either an empty string or a resource ID of 0.
3. The help message when your application dims the menu item. This message is displayed either when the item itself is not specified in a menu-item component of this `'hmnu'` resource or when its help message *is* specified in a menu-item component, but specified with either an empty string or a resource ID of 0.
4. The help message when a menu item is enabled and checked. This message is displayed either when the item itself is not specified in a menu-item component of this `'hmnu'` resource or when its help message *is* specified in a menu-item component, but specified with either an empty string or a resource ID of 0.
5. The help message when a menu item is enabled and marked (with a character other than a checkmark). This message is displayed either when the item itself is not specified in a menu-item component of this `'hmnu'` resource or when its help message *is* specified in a menu-item component, but specified with either an empty string or a resource ID of 0.

For missing items (as for the rest of the items listed in an `'hmnu'` resource), you store the help messages in text strings within this resource or in separate `'STR '`, `'STR#'`, `'PICT'`, or `'TEXT'` and `'styl'` resources. For the first element in the missing-items component, use one of the identifiers described in "Specifying the Format for Help Messages" on page 3-23. These identifiers indicate how and where you store your help messages. Then, depending on the identifier you specify, for the next four elements supply either text strings for help messages or resource IDs of resources that contain help messages.

There are two additional identifiers that you can specify for menu items in `'hmnu'` resources. These identifiers are explained in "Specifying Help for a Changing Menu Item" on page 3-43 and in "Specifying Resources by Item Name" on page 3-45.

Identifier	Purpose
`HMCompareItem`	The Help Manager displays help for the current menu item only when it matches a specified string.
`HMNamedResourceItem`	The Help Manager displays the help message from the resource that has the same name as the current menu item.

Listing 3-2 on page 3-35 illustrates the help resource for a menu titled Colors. Notice in the missing-items component that the element describing dimmed states for menu items has the message "Not available; either you have not selected text to color, or your monitor does not support color." Because this resource doesn't specify a message for any individual command's dimmed state, this message appears in help balloons for the Blue, Green, and Red commands whenever the application disables them. If there are many reasons why your application may have dimmed an item, don't name them all. Instead, describe one or two of the most likely reasons.

Note

As described in the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*, system software automatically dims your application's menus as appropriate whenever an alert box or a fixed-position modal dialog box appears on the screen. You supply help messages for menu titles and menu items dimmed by system software in the third and fourth elements of the menu-title component of the `'hmnu'` resource, as described in the next section. If your application uses *movable* modal dialog boxes or modeless dialog boxes, your application must dim its menus as appropriate and provide an alternate `'hmnu'` resource for this state, as described in "Providing Help Balloons for Menus You Disable for Dialog Boxes" beginning on page 3-47. ◆

Listing 3-2 Rez input for the missing-items component of an `'hmnu'` resource

```
resource 'hmnu' (132, "Colors", purgeable) {
   /*header component*/
   HelpMgrVersion, hmDefaultOptions, 0, 0,
   /*missing-items component*/
   HMStringItem {
      "",      /*no missing enabled items*/
      /*help messages for all items that app dims are below*/
      "Not available; either you have not selected "
         "text to color, or your monitor does "
         "not support color.",
      "",      /*no missing enabled-and-checked items*/
      "",      /*no missing enabled-and-marked items*/
      },
   {  /*menu-title component*/
      HMStringItem {    /*use these P-strings for help messages*/
         /*use string below when menu is enabled*/
         "Colors menu\n\nUse this menu to display text in color.",
         /*use string below when app dims menu title*/
         "Colors menu\n\nUse this menu to display text in color."
            "Not available because this monitor does not
            support color.",
         /*use string below when system software dims menu */
         /* title for an alert or modal dialog box*/
         "Colors menu\n\nUse this menu to display text in color. "
            "Not available because a dialog box is on the "
            "screen.",
         /*use string below for all items when system dims */
         /* them for alert and modal dialog boxes*/
         "Colors your selected text. This item is not "
            "available because a dialog box is on the screen.",
      },
      /*first menu-item component: Blue command*/
      HMStringItem {/*use these P-strings for help messages*/
         /*use string below when command is enabled*/
         "Displays the selected text in blue.",
         "",   /*use missing-items help when app dims menu*/
         "",   /*can't check command, so use empty string here*/
         "",   /*can't mark command, so use empty string here*/
      },
      /*second menu-item component: Green command*/
      HMStringItem {/*use these P-strings for help messages*/
         /*use string below when command is enabled*/
         "Displays the selected text in green.",
         "",   /*use missing-items help when app dims menu*/
         "",   /*can't check command, so use empty string here*/
         "",   /*can't mark command, so use empty string here*/
      },
      /*third menu-item component: Red command*/
      HMStringItem {/*use these P-strings for help messages*/
         /*use string below when command is enabled*/
         "Displays the selected text in red.",
         "",   /*use missing-items help when app dims menu*/
         "",   /*can't check command, so use empty string here*/
         "",   /*can't mark command, so use empty string here*/
      }
   }
};
```

Specifying Help for Menu Titles and for Items Dimmed by System Software

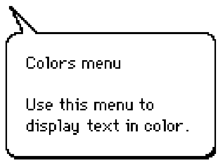
After the missing-items component, use the menu-title component to specify the help messages for the menu title and for menu items dimmed by system software. The menu-title component consists of the following five elements:

1. An identifier (either `HMStringItem`, `HMSTRResItem`, `HMStringResItem`, `HMPictItem`, `HMTEResItem`, or `HMSkipItem`) that indicates the format of the help messages.
2. The help message for the menu title when the menu title is enabled.
3. The help message for the menu title when your application dims the menu title.
4. The help message for the menu title when system software dims the menu title at the appearance of an alert box or a modal dialog box.
5. The help message for all menu items when system software dims them at the appearance of an alert box or a modal dialog box.

As in the missing-items component, use the first element in the menu-title component to specify an identifier that describes the format for the help messages. Depending on the identifier you specify, for the other elements you supply either text strings for the help messages or the resource IDs of resources that contain the help messages. The second, third, and fourth elements correspond to states of the menu title; the fifth element corresponds to a state of all the menu items.

Use the second element of the menu-title component to specify a help message that describes the purpose of the menu when it's enabled. For menus in the menu bar, the beginning of the message should name the menu and then concisely describe what kinds of commands are in the menu, as shown in Figure 3-10. (Listing 3-2 on page 3-35 specifies the menu title—"Colors menu"—and then includes the special characters `\n\n` to specify two new lines in a Rez input file before specifying a description of the menu itself.)

Figure 3-10 A help balloon for an enabled menu title

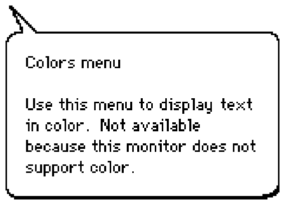

Colors menu

Use this menu to display text in color.

Because some pull-down menus in the menu bar are identified by icons, not words, offer additional clarification in your help balloon by always providing a name for pull-down menus. For pop-up menus, simply describe what the user does with the menu; don't give the menu a name.

Use the third element of the menu-title component to specify a help message that identifies the menu, describes what it does, and then describes why your application has dimmed the menu title. As much as possible, repeat the text that you use for the title's enabled state, then describe why it is not enabled. See Figure 3-11 for an example and see Listing 3-2 on page 3-35 for the Rez input that specifies the help message for the help balloon shown in the figure.

Figure 3-11 A help balloon for a dimmed menu title

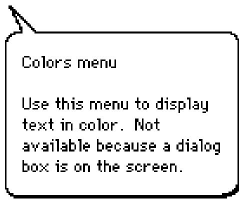

Colors menu

Use this menu to display text in color. Not available because this monitor does not support color.

In general, you should use the phrase "Not available because" to introduce your explanation of a dimmed title. If there are several reasons why a menu title might be dimmed, don't name them all. Instead, describe one or two of the most likely reasons.

Use the fourth element of the menu-title component to specify a help message that describes why system software has dimmed the menu title—that is, because the user must respond to an alert box or modal dialog box on the screen. Figure 3-12 illustrates an appropriate help balloon for this situation.

Figure 3-12 A help balloon for a menu title dimmed by the Dialog Manager


Colors menu

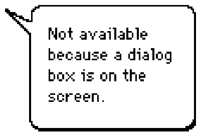
Use this menu to display text in color. Not available because a dialog box is on the screen.

Starting with system software version 7.0, users have been able to use selected menus while the screen displays an alert box or a modal dialog box. For example, the Show Balloons (or Hide Balloons) command is always available from the Help menu so that users can see your help balloons for the modal dialog box or alert box. While some menus are accessible (in particular, the Help, Keyboard, and—when appropriate—Edit menus), others aren't. The chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* describes the circumstances under which menus are enabled or disabled when alert boxes and dialog boxes are displayed.

Note
If your application uses *movable* modal dialog boxes, you must dim your menus and provide an alternate `'hmnu'` resource for this state, as described in "Providing Help Balloons for Menus You Disable for Dialog Boxes" beginning on page 3-47. ◆

Use the fifth element to specify the help message for all menu items whenever system software dims them because of an alert box or a modal dialog box. Because this message is used for all items in the menu, it needs to be somewhat general, as shown in Figure 3-13.

Figure 3-13 A help balloon for menu items dimmed by the Dialog Manager



Specifying Help for Menu Items

After you create the header component, the missing-items component, and the menu-title component, you specify help messages in a menu-item component for each menu item. The menu-item components in the `'hmnu'` resource must appear in the order in which their corresponding menu items appear in the menu. (Because of this, if you have menu items that your application can add while it is running, you should add these dynamic items to the end of the menu to simplify your implementation of help balloons for your nondynamic items.)

The menu-item component consists of the following five elements:

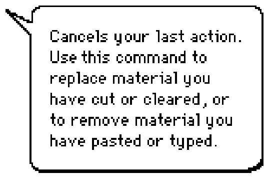
1. An identifier (either `HMStringItem`, `HMSTRResItem`, `HMStringResItem`, `HMPictItem`, `HMTEResItem`, or `HMSkipItem`) that indicates the format of the help messages.
2. The help message for the menu item when the item is enabled.
3. The help message for the menu item when your application dims the item.
4. The help message for the menu item when the item is enabled and checked.
5. The help message for the menu item when the item is enabled and marked with a character other than a checkmark.

For the first element of each menu-item component, supply an identifier to describe the format of the help messages in that component. Then, depending on the identifier you specify, use the other elements of the menu-item component to supply either text strings for the help messages or the resource IDs of resources that contain the help messages.

You can use the `HMSkipItem` identifier for items that appear in your menu but for which you don't provide a help balloon. For example, you can specify `HMSkipItem` for divider lines that appear in menus. (Divider lines cannot have help balloons.) If you specify `HMSkipItem`, the Help Manager does not display help balloons for that menu item, even if the missing-items component specifies a help message.

For the second element, specify a help message that describes what the item usually does. Don't name the menu item. Begin with a verb describing what happens when the user chooses the item. For example, the help balloon for the Undo command in an Edit menu should contain something similar to the information in Figure 3-14.

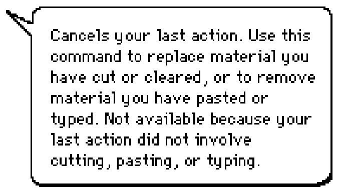
Figure 3-14 A help balloon for a menu item


Cancels your last action. Use this command to replace material you have cut or cleared, or to remove material you have pasted or typed.

For menu items that display a dialog box, it is usually unnecessary to state that a dialog box will appear. The fact that a menu item displays a dialog box is not what the user wants to know; the user wants to know what choosing the menu item ultimately accomplishes.

For the third element, specify a help message that describes what the menu item does and why your application has dimmed the item. As much as possible, repeat the text that you use for the item's enabled state, and then describe why it is not enabled. In general, you should use the phrase "Not available because" to introduce your explanation of the dimmed item. If there are multiple reasons why an item might be dimmed, don't name them all. Instead, describe one or two of the most likely reasons. Figure 3-15 gives an example of a help message for a dimmed Undo command.

Figure 3-15 A help balloon for a dimmed menu item



Cancels your last action. Use this command to replace material you have cut or cleared, or to remove material you have pasted or typed. Not available because your last action did not involve cutting, pasting, or typing.

If your application checks or otherwise marks a menu item, use the fourth and fifth elements of that item's component in the `'hmnu'` resource to describe the special condition indicated by that state. As with dimmed states, try to repeat the text that you use for the title's enabled state, and then describe why it is checked or marked. If there are multiple reasons why an item might be checked or marked, don't name them all. Instead, describe one or two of the most likely reasons.

Note that, for any component in the resource, you can specify only one format for all of its help messages. For example, if you specify the `HMSTRResItem` identifier in a menu-item component for the Undo command, you must store all help messages specified in that component in `'STR '` resources. (However, if you specify a resource ID of 0 or an empty string as the help message of any item in order to use the help message from the missing-items component, the help message follows the format specified in the missing-items component.)

You do not have to provide a help message for every state of a menu item. If you do not provide a help message for a particular state, the Help Manager uses the help message specified in the missing-items component. If the missing-items component does not specify a help message for that state either, the Help Manager does not display a help balloon.

Listing 3-3 shows a sample 'hmnu' resource for another Edit menu.
Although Listing 3-1 and Listing 3-2 illustrate 'hmnu' resources that contain their own Pascal-string help messages, you should keep your help messages in separate, more easily localized resources. The 'hmnu' resource in Listing 3-3 stores its help messages in a separate 'STR#' resource (which is given a corresponding resource ID of 130 for easier maintenance).

Listing 3-3 Rez input for corresponding 'hmnu' and 'STR#' resources

```
resource 'hmnu' (130, "Edit menu help", purgeable) {
   HelpMgrVersion, 0, 0, 0,    /*standard header component*/
   HMSkipItem {                /*missing-items component*/
      /*no missing items, so skip to menu-title component*/
      },
   {  /*menu title and items below*/
      /*menu-title component*/
      HMStringResItem {/*use an 'STR#' for help messages*/
         130,1,   /*'STR#' res ID, index when menu is enabled*/
         130,2,   /*'STR#' res ID, index when app dims menu*/
         130,3,   /*'STR#', index for title that system */
                  /* software dims for all alert and modal */
                  /* dialog boxes*/
         130,4    /*'STR#', index for items that system */
                  /* software dims for all alert and modal */
                  /* dialog boxes*/
      },
         /*first menu-item component: Undo command*/
      HMStringResItem { /*use 'STR#' resource for help messages*/
         130,5,   /*'STR#' res ID, index when item is enabled*/
         130,6,   /*'STR#' res ID, index when item is dimmed*/
         0,0,     /*can't check command*/
         0,0      /*can't mark command*/
      },

         /*second menu-item component: divider line*/
      HMSkipItem {   /*no balloon help for divider lines*/
      },
         /*third menu-item component: Cut command*/
      HMStringResItem { /*use an 'STR#' for help messages*/
         130,7,   /*'STR#' res ID, index when item is enabled*/
         130,8,   /*'STR#' res ID, index when app dims item*/
         0,0,     /*can't check command*/
         0,0      /*can't mark command*/
      },
         /*menu-item component for Copy command goes here*/
   }
};

resource 'STR#' (130, "Edit menu help strings") {
   /*help text for Edit menu*/
   { /*array StringArray: 17 elements*/
   /*[1] help text for enabled Edit menu title*/
   "Use this menu to cancel your last action, to manipulate "
      "text, to select the entire content of a document, "
      "and to show what's on the Clipboard.";
   /*[2] help text for Edit menu title dimmed by app*/
   "Use this menu to cancel your last action, to manipulate "
      "text, to select the entire content of a document, "
      "and to show what's on the Clipboard. Not "
      "available now.";
   /*[3] help text for Edit menu title dimmed by system */
   /* software for all alert and modal dialog boxes */
   /* that don't contain editable text items*/
   "Use this menu to cancel your last action, to manipulate "
      "text, to select the entire content of a document, and "
      "to show what's on the Clipboard. Not available "
      "because a dialog box is on the screen.";
   /*[4] help for Edit menu items that system software dims */
   /* for all alert and modal dialog boxes */
   /* that don't contain editable text items*/
   "Not available because a dialog box is on the screen.";
   /*[5] help text for enabled Undo command*/
   "Cancels your last action. Use this command to replace "
      "material you have cut or cleared, or to remove material "
      "you have pasted or typed.";
   /*[6] help text for Undo command dimmed by app*/
   "Cancels your last action. Use this command to replace "
      "material you have cut or cleared, or to remove material "
      "you have pasted or typed. Not available because your "
      "last action did not involve cutting, pasting, "
      "or typing.";
   /*help text for all other commands goes here*/
   }
};
```

The 'hmnu' resource in Listing 3-3 specifies the standard balloon definition function and variation code in the third and fourth elements of the header component. The missing-items component is specified using the HMSkipItem identifier, meaning that this 'hmnu' resource does not provide any help balloons for menu items that are missing from this resource or that do not have help messages specified for any states.

Following the menu-title component, the menu-item components for the menu items are listed in the order in which the items appear in the menu. For menu-item components that do not specify information for a particular state, the Help Manager normally uses the information from the missing-items component. However, in Listing 3-3 the 'hmnu' resource does not specify a help message in the missing-items component. Instead, all help messages are specified in each menu-item component in this resource. Because there are no enabled-and-checked or enabled-and-marked states for the Undo and Copy commands, these states are specified with resource IDs of 0.

As described in the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*, system software does not dim your application's Edit menu when you display a dialog box that contains editable text items. Listing 3-3 nevertheless provides help messages for a dimmed Edit menu for those instances when the application displays alert boxes and dialog boxes that do not contain editable text items.

Specifying Help for a Changing Menu Item

If you have a menu item that changes names, you can use the HMCompareItem identifier to compare a string against the current menu item in that position. If the string specified after the HMCompareItem identifier matches the name of the current menu item, the Help Manager displays the help messages specified in the next four elements of the 'hmnu' resource. Because of performance considerations, the HMCompareItem identifier shouldn't be used unless necessary.

A menu-item component that uses the HMCompareItem identifier uses a different format than other menu-item components do. Here are the seven elements you use for specifying help in an 'hmnu' resource for a changing menu item.

1. The HMCompareItem identifier.
2. The string to compare against current menu item.
3. The identifier (either HMStringItem, HMSTRResItem, HMStringResItem, HMPictItem, HMTEResItem, or HMSkipItem) that indicates the format of the help messages.
4. The help message for the menu item when the item is enabled.
5. The help message for the menu item when the item is dimmed.
6. The help message for the menu item when the item is enabled and checked.
7. The help message for the menu item when the item is enabled and marked.

Create a component that uses the `HMCompareItem` identifier for every item name that can appear in a particular menu position. For example, Listing 3-4 shows an `'hmnu'` resource for a menu command that toggles between Show Colors and Hide Colors.

Note

It is important to provide components for all possible strings that can appear in the changing item. Also note that if you use the `HMCompareItem` identifier for a menu item, you should ensure that the following menu item has a text string that is different from the one that the `HMCompareItem` menu-item component compares against. ◆

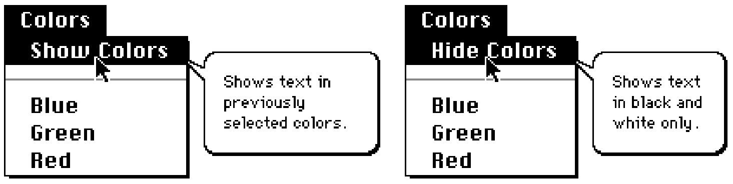
Listing 3-4 Rez input for an `'hmnu'` resource that uses `HMCompareItem` for a changing menu item

```
resource 'hmnu' (132, "Colors menu help", purgeable) {
   /*see Listing 3-2 for missing-items example*/
   /*see Listing 3-2 for Colors menu title's help example*/
      HMCompareItem {   /*help message if first command is */
                        /* called Show Colors*/
         "Show Colors",
         HMStringResItem {
            132, 1,        /*enabled*/
            0, 0,          /*use missing items*/
            0, 0,          /*item can't be checked*/
            0, 0           /*no marked state*/
         },
      },
      HMCompareItem {   /*help if the first command is */
                        /* called Hide Colors*/
         "Hide Colors",
         HMStringResItem {
            132, 2,        /*enabled*/
            0, 0,          /*use missing items*/
            0, 0,          /*item can't be checked*/
            0, 0           /*no marked state*/
         },
      },
      /*Blue command's help messages*/
      HMStringItem { /*use these P-strings for help messages*/
            /*use string below when command is enabled*/
         "Displays the selected text in blue.",
         "",   /*use missing-items help when menu is dimmed*/
         "",   /*can't check command--use empty string here*/
         "",   /*can't mark command--use empty string here*/
      },
      /*see Listing 3-2 for other commands' examples*/
   }
};

resource 'STR#' (132, "Hide & Show Colors commands help text") {
      {
      /*[1] help text for enabled Show command*/
      "Shows text in previously selected colors.";
      /*[2] help text for enabled Hide command*/
      "Shows text in black and white only.";
      }
};
```

As illustrated in Figure 3-16, when the menu command is Show Colors, the Help Manager displays the help message described by the first `HMCompareItem` component. When the menu command is Hide Colors, the Help Manager displays the help message described by the second `HMCompareItem` identifier.

Figure 3-16 Help balloons for a changing menu item


Colors
Show Colors
Blue
Green
Red
Shows text in previously selected colors.
Colors
Hide Colors
Blue
Green
Red
Shows text in black and white only.

Specifying Resources by Item Name

You can also specify help messages in a component with the `HMNamedResourceItem` identifier, which causes the Help Manager to use a resource whose name matches the name and state of the current menu item. A menu-item component that uses the `HMNamedResourceItem` identifier uses a different format than the other menu-item components do. Here are the two elements you use for specifying named resources as help messages in an `'hmnu'` resource.

1. The `HMNamedResourceItem` identifier.
2. A resource type (either `'STR '`, `'PICT'`, or, for text, `'TEXT'`) that contains the help messages for the current menu item. If you specify `'TEXT'`, you also get style information for the `'TEXT'` resource by creating a resource of type `'styl'` with the same name.

To provide help for a menu item when it is enabled, create a resource with the same name as the menu item, then put the help message for the enabled menu item in this resource. The Help Manager uses the `GetNamedResource` function to find the resource—of the type specified in the second element of the menu-item component—that has the same name as the current menu item.

To provide help for a menu item when it is dim, create a resource with a name consisting of the menu item and an exclamation point (!), then put the help message for the dimmed menu item in this resource. When the menu item is dimmed, the Help Manager appends an exclamation point to the name of the menu item and searches for a resource by that name. Similarly, if a menu item is enabled and marked with a checkmark or other mark, the Help Manager appends the mark to the name of the current menu item and looks for a resource by that name.

For example, the `'hmnu'` resource in Listing 3-5 specifies that the Help Manager extracts the help message from a resource named `Red` of type `'STR '` when displaying a help balloon for an enabled menu command named Red. If the menu item is dimmed, the Help Manager gets the `'STR '` resource with the name `Red!` and uses its text string for the help message. If the Red command could be marked with an asterisk (*), the Help Manager would search for the resource with the name `Red*` of type `'STR '`.

Listing 3-5 Rez input for specifying help messages with named resources

```
resource 'hmnu' (132, "Colors menu help", purgeable) {
         /*see Listing 3-2 for header, missing-items, */
         /* menu-title, and menu-item components*/
      HMNamedResourceItem {   /*Red command's help message*/
         'STR '/*use the 'STR ' resource named "Red"*/
      }
   }
};
resource 'STR ' (333, "Red") {   /*help text for enabled */
                                 /* Red command*/
      "Displays the selected text in red."
};
resource 'STR ' (334, "Red!") {  /*help text for dimmed */
                                 /* Red command*/
      "Not available; either you have not selected text to "
      "color, or your monitor does not support color.",
};
```

Providing Help Balloons for Menus You Disable for Dialog Boxes

The Dialog Manager and the Menu Manager interact to provide various degrees of access to the menus in your menu bar. For alert boxes and modal dialog boxes without editable text items, you can simply allow system software to dim your menu titles and menu items as appropriate. As described in "Specifying Help for Menu Titles and for Items Dimmed by System Software" beginning on page 3-36, you specify help balloons for these dimmed menu titles and menu items in the fourth and fifth elements of your `'hmnu'` resources' menu-title components.

However, because system software cannot handle the Undo or Clear command (or any other context-appropriate command) for you, your application should handle its own menu bar access for modal dialog boxes with editable text items by performing the following tasks:

- Use the Menu Manager function `DisableItem` to disable the Apple menu or the first item in the Apple menu (typically, your application's About command) in order to take control of its menu bar access when displaying a modal dialog box.
- Use the Menu Manager function `DisableItem` to disable all of your application's menus except the Edit menu, as well as any inappropriate commands in the Edit menu.
- Use the Dialog Manager procedures `DialogCut`, `DialogCopy`, `DialogPaste`, and `DialogDelete` to support the Cut, Copy, Paste, and Clear commands in editable text items.
- Provide your own code for supporting the Undo command.
- Use the Menu Manager function `EnableItem` to enable your application's items in the Help menu as appropriate (system software disables all items except the Hide Balloons / Show Balloons command).

You don't need to do anything else for the system-handled menus—namely, Application, Keyboard, and Help. System software handles these menus for you automatically.

Although it always leaves the Help, Keyboard, and Application menus and their commands enabled, system software does nothing else to manage the menu bar when you display movable modal and modeless dialog boxes. Instead, your application should allow or deny access to the rest of your menus as appropriate to the context. For example, if your application displays a modeless dialog box for a search-and-replace command, you should allow access to the Edit menu to assist the user with the editable text items, and you should allow use of the File menu so that the user can open another file to be searched. However, you should disable other menus if their commands cannot be used inside the active modeless dialog box.

When creating a modeless dialog box, your application should perform the following tasks:

- Use the Menu Manager function `DisableItem` to disable only those menus whose commands are invalid in the current context.
- If the modeless dialog box includes editable text items, use the Dialog Manager procedures `DialogCut`, `DialogCopy`, `DialogPaste`, and `DialogDelete` to support the Cut, Copy, Paste, and Clear commands in editable text items.
- Enable your application's items in the Help menu, as appropriate. (System software disables all items except the Hide Balloons / Show Balloons command.)

When your application creates a movable modal dialog box, it should perform the following tasks:

- Leave the Apple menu enabled so that the user can open other applications with it.
- If your movable modal dialog box contains editable text items, leave the Edit menu enabled but use the Dialog Manager procedures `DialogCut`, `DialogCopy`, `DialogPaste`, and `DialogDelete` to support the Cut, Copy, Paste, and Clear commands.
- Use the Menu Manager function `DisableItem` to disable all of your other menus.

See the chapters "Menu Manager" and "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for more information about menus, alert boxes, and dialog boxes.

When you use the Menu Manager function `DisableItem` to dim your menus, the Help Manager does not know that you have dimmed them because you're displaying a dialog box; instead, the Help Manager assumes that you've dimmed them for some other reason. Whenever you dim your own menus—for whatever reason—the Help Manager always uses the second element of your menu-title component and the second elements of your menu-item components.

To provide help messages that explain that the menu and its items are dim because your application dimmed them to display a dialog box, you must use an alternate `'hmnu'` resource.

For example, if an application were to allow only one document at a time to be opened, it would dim the New command in the File menu whenever a document were open. The second element of that item's component specifies a help message similar to this.

```
/*help message for dimmed New command*/
"Opens a new SurfWriter document called \"Untitled\". "
"Not available now because a SurfWriter document is "
"already open. (SurfWriter can open only one document "
"at a time.)";
```

This is not an appropriate message for the item's help balloon when the application displays a modal dialog box that contains an editable text item—but unless the application changes the `'hmnu'` resource for its File menu, this is the message that the Help Manager displays.

To handle those menus that you dim for dialog boxes, your application must use alternate 'hmnu' resources. In an alternate 'hmnu' resource, use the second element of the missing-items component and the second element of the menu-title component to specify help balloons for the menu's dimmed title and all of its dimmed items, as shown in Listing 3-6.

Listing 3-6 Specifying an alternate 'hmnu' resource for a menu that your application disables when it displays movable modal dialog boxes

```
resource 'hmnu' (kFileHelpID, purgeable)
{  /*use this when my application dims the menu to display */
   /* a modal dialog box with editable text items*/
   /*header component*/
   HelpMgrVersion, hmDefaultOptions, 0, 0,
   /*missing-items component*/
   HMStringResItem {
      0, 0,          /*missing enabled items: not applicable */
                     /* because they're all dim*/
      256, 1,        /*use this help string for all dimmed */
                     /* menu items--they're all missing from */
                     /* this resource*/
      0, 0,          /*missing enabled-and-checked items: not */
                     /* applicable because they'd be dimmed*/
      0, 0,          /*missing enabled-and-marked items: not */
                     /* applicable because they'd be dimmed*/
   },
   /*menu-title component*/
   {  /*File menu title help when dimmed for a movable modal*/
      HMStringResItem {
         0, 0,          /*no enabled title: it's dimmed*/
         256, 2,        /*use this help string for menu title */
                        /* dimmed for a movable modal dialog*/
         0, 0,          /*Help Manager doesn't look here for */
                        /* movable modal dialogs*/
         0, 0,          /*Help Manager doesn't look here for*/
                        /* movable modal dialogs*/
      },
   }        /*use missing-items info for all dimmed menu items*/
};
resource 'STR#' (256, "help messages for dimmed menus") {
/*use these when my application dims menus to show a */
/* modal dialog box*/
   {
   /*[1] Dimmed items help text*/
   "Not available now because a dialog box is on "
   "the screen.".
   /*[2] help message for dimmed File menu title*/
   "File menu\n\nUse this menu to open, close, save, and print "
   "SurfWriter documents, and to quit SurfWriter. "
   "Not available because a dialog box is on the screen.";
   /*[3] help message for dimmed Tools menu title*/
   "Tools menu\n\nUse this menu to ... " /*more text goes here*/
      "Not available because a dialog box is on the screen.";
   /*help messages for other dimmed menu titles go here*/
};
```

Use the `HMSetMenuResID` function to associate alternate `'hmnu'` resources with your menus whenever your application displays a movable modal dialog box. Listing 3-7 illustrates how an application disables its menus and then reassigns them appropriately, using alternate `'hmnu'` resources before displaying a dialog box.

Listing 3-7 Reassigning `'hmnu'` resources before displaying a movable modal dialog box

```
PROCEDURE MyAdjustMenusForDialogs;
VAR
   window:     WindowPtr;
   windowType: Integer;
   myErr:      OSErr;
   menu:       MenuHandle;
BEGIN
   window := FrontWindow;
   windowType := MyGetWindowType(window);
   CASE windowType OF
      kMyModalDialogs:
      BEGIN
         menu := GetMenuHandle(mApple);   {get handle to Apple menu}
         IF menu = NIL THEN
            EXIT(MyAdjustMenusForDialogs);
         DisableItem(menu, 0);   {disable Apple menu to get control of menus}
         myErr := HMSetMenuResID(mFile, kFileHelpID); {set up help balloons}
         menu := GetMenuHandle(mFile);       {get handle to File menu}
         IF menu = NIL THEN
            EXIT(MyAdjustMenusForDialogs);
         DisableItem(menu, 0);      {disable File menu}
         myErr := HMSetMenuResID(mFile, kFileHelpID); {set up help balloons}
         IF myErr <> NoErr THEN
            EXIT(MyAdjustMenusForDialogs);
         menu := GetMenuHandle(mTools);   {get handle to Tools menu}
         IF menu = NIL THEN
            EXIT(MyAdjustMenusForDialogs);
         DisableItem(menu, 0);         {disable Tools menu}
         myErr := HMSetMenuResID(mTools, kToolsHelpID);  {help balloons}
         IF myErr <> NoErr THEN
            EXIT(MyAdjustMenusForDialogs);
         MyAdjustEditMenuForModalDialogs;
      END;        {of kMyModalDialogs CASE}
      kMyGlobalChangesModelessDialog:
         ;  {adjust menus here as needed}
      kMyMovableModalDialog:
         ;  {adjust menus here as follows: }
            { diable all menus except Apple, }
            { call MyAdjustEditMenuForModalDialogs for editable text items}
   END; {of CASE}
END;
```

The MyAdjustMenusForDialogs routine in Listing 3-7 first determines what type of dialog box is in front: modal, movable modal, or modeless. For modal dialog boxes, MyAdjustMenusForDialogs disables the Apple menu so that the application can take control of its menus away from the Dialog Manager. The MyAdjustMenusForDialogs routine then uses the Menu Manager routines GetMenuHandle and DisableItem to disable all other application menus except the Edit menu.

To adjust the items in the Edit menu, MyAdjustMenusForDialogs calls another application-defined routine, MyAdjustEditMenuForModalDialogs. The MyAdjustEditMenuForModalDialogs routine, which is not shown in this volume, uses application-defined code to implement the Undo command; uses the Menu Manager procedure EnableItem to enable the Cut, Copy, Paste, and Clear commands when appropriate; and disables the commands that support Edition Manager capabilities.

After removing a dialog box from the screen, enable the appropriate menus again and use the HMSetMenuResID function to reassociate your original help balloons with the reenabled menus. (You can pass –1 to the HMSetMenuResID function to remap menus to their previous 'hmnu' resources.)

Providing Help Balloons for Items in Dialog Boxes and Alert Boxes

For dialog boxes and alert boxes defined with item list ('DITL') resources, you can provide help balloons for individual items in the dialog box or alert box by supplying a resource of type 'hdlg' (dialog-item help). When an item has a help balloon associated with it, the Help Manager automatically displays and removes the help balloon as the user moves the cursor into and out of the item's display rectangle. The Help Manager can display different help balloons for various states of an item—by highlight value if the item is a control, and by enabled and disabled states for items that are not controls.

Note
BalloonWriter, available from APDA, gives nonprogrammers an easy, intuitive way to create help balloons for dialog and alert boxes. BalloonWriter creates `'hdlg'` resources as appropriate and places them in the resource file of your application; BalloonWriter likewise creates and stores `'STR '`, `'STR#'`, and `'TEXT'` resources that contain the help messages authored by nonprogramming writers. ◆

You can also provide help balloons for other areas of a dialog box or alert box using the `'hwin'` (window help) resource as described in "Providing Help Balloons for Static Windows" on page 3-65.

To create help balloons for items in dialog boxes or alert boxes, create an `'hdlg'` resource that corresponds to an item list resource. You associate the information defined in the `'hdlg'` resource to the alert or dialog box in one of three ways:

- by adding an item of type `HelpItem` to the item list resource
- by supplying a resource of type `'hwin'`
- by calling the `HMScanTemplateItems` function from your application

The `'hdlg'` resource specifies the tip, the alternate rectangle, and help messages for items in a dialog box or alert box. The item list resource describes the items, and, if it includes an item of type `HelpItem`, it can contain the resource ID of a corresponding `'hdlg'` resource. The Help Manager uses the display rectangles defined in the item list resource as the hot rectangles for the items. The Help Manager uses the alternate rectangles specified in the `'hdlg'` resource for transposing help balloons' tips when trying to fit the balloons onscreen.

For those items designated in the `'hdlg'` resource, the Help Manager automatically tracks the cursor and displays help balloons when the following conditions are met: the dialog or alert box has an item of type `HelpItem` in its item list resource; your application calls the Dialog Manager routine `ModalDialog`, `IsDialogEvent`, `NoteAlert`, `StopAlert`, `CautionAlert`, or `Alert`; and help is enabled.

If the cursor passes over any active windows, including dialog or alert boxes, the Help Manager searches the current resource file for resources of type `'hwin'` (described in "Providing Help Balloons for Static Windows" on page 3-65). The Help Manager attempts to match either the title of the window or the `windowKind` value in its window record with the title or `windowKind` value specified in an `'hwin'` resource. The matched `'hwin'` resource, in turn, specifies the resource ID of an `'hdlg'` or `'hrct'` (rectangle help) resource that contains the relevant help message. (The `'hrct'` resource is described in "Providing Help Balloons for Static Windows" on page 3-65.) As described in "Providing Help Balloons for Window Content" on page 3-63, the `'hwin'` resource can provide help for various other interface features across the entire window as well as for items in a dialog box or an alert box.

If you prefer, you can track and display help balloons for modal dialog boxes and alert boxes yourself by using an event-filter function and calling the `HMScanTemplateItems` function. Using `HMScanTemplateItems` requires you to modify your code. For further information on `HMScanTemplateItems`, see “Setting and Getting Information for Help Resources” beginning on page 3-114.

As shown here, a Rez input file for an `'hdlg'` resource contains a header component, a missing-items component, and dialog-item components.

Component	Element
Header	Help Manager version
	Index number of starting item (first item is number 0)
	Options
	Balloon definition function
	Variation code
Missing items	Tip’s coordinates
	Alternate rectangle
	Identifier for help messages
	Help message for missing, unselected active controls (that is, those with highlight values of 0), or for missing enabled items that are not controls
	Help message for missing dimmed controls (that is, those with highlight values of 255), or for missing disabled items that are not controls
	Help message for missing active controls that are “on” (that is, those with highlight values of 1)
	Help message for missing active controls with highlight values other than 0, 1, and 255
First dialog item	Tip’s coordinates
	Alternate rectangle
	Identifier for help messages
	Help message for an active, unselected control (that is, one with a highlight value of 0), or for an enabled item that is not a control
	Help message for a dimmed control (that is, one with a highlight values of 255), or for a disabled item that is not a control
	Help message for an active control that is “on” (that is, one with a highlight value of 1)
	Help message for an active control with a highlight value other than 0, 1, and 255
Next dialog item	(Same as for first dialog item)
	.
	.
	.
Last dialog item	(Same as for first dialog item)

As described in greater detail later, the way the Help Manager interprets many of the elements depends on whether the item it describes is a control, such as a checkbox or radio button, or something else, such as static text or an icon.

Specifying Header Information for the 'hdlg' Resource

Use the header component to specify the Help Manager version number, the starting index, options, the balloon definition function, and the variation code. As in the other help resources, specify the `HelpMgrVersion` constant for the first element of the header component of the `'hdlg'` resource.

You use the second element to associate the help messages beginning at any item number and then continuing sequentially through the item list (`'DITL'`) resource. To derive an item number to start from, the Help Manager adds the index number you specify for this element to the number of the first item in the item list resource. Thus, index number 0 starts with the item number 1 in the item list resource (because 0 plus 1 equals 1). For example, to describe help messages for only the fifth through seventh items, specify 4 as the starting index in the header component and, because 4 plus 1 equals 5, provide help messages that start with the fifth and proceed through the sixth and seventh items.

For the options element, specify a constant (normally, `hmDefaultOptions`) or the sum of several constants' values from this list. (These options are described in "Specifying Options in Help Resources" beginning on page 3-25.)

```
CONST hmDefaultOptions     = 0;  {use defaults}
      hmUseSubID           = 1;  {use subrange resource IDs }
                                 { for owned resources}
      hmAbsoluteCoords     = 2;  {ignore coords of window }
                                 { origin and treat upper-left }
                                 { corner of window as 0,0}
      hmSaveBitsNoWindow   = 4;  {don't create window; save }
                                 { bits; no update event}
      hmSaveBitsWindow     = 8;  {save bits behind window }
                                 { and generate update event}
```

Specify the balloon definition function and variation code (both typically 0) in the fourth and fifth elements of the header component. (These are described in detail in "Specifying Header Information for the 'hmnu' Resource" beginning on page 3-32.)

Specifying Missing-Item Information

Following the header component, you can specify the help message for items that are missing from the `'hdlg'` resource or that are present but have no help messages defined for a particular state. (The function of the missing-items component of the `'hdlg'` resource is similar to that of the missing-items component of the `'hmnu'` resource. For details, see "Specifying Help for Menu Items Missing From the Resource" beginning on page 3-33.)

In the missing-items component, use the first element to specify a set of tip coordinates and use the second element to specify an alternate rectangle. Both specifications apply to the help messages specified in the other elements of this component.

The tip's coordinates are always relative to the item's position in the dialog box. If you specify the point (0,0) as a default tip, then it is placed 10 pixels from the right and 10 pixels from the bottom of the item's rectangle (as specified in the item list resource) for all missing items. To move the missing item's tip relative to this default location, you can specify positive or negative integers in place of the coordinates (0,0).

If you want an alternate rectangle that is either larger or smaller than a display rectangle, use the missing item's alternate rectangle to specify offsets that apply to the display rectangles for all items in the dialog box. (Remember that the alternate rectangle is used by the Help Manager for transposing the tip if a help balloon does not fit onscreen.) The Help Manager adds the top, left, bottom, and right offsets to the coordinates of an item's display rectangle. For example, if you specify (0,0,0,0) as the missing item's alternate rectangle offsets, the Help Manager uses the display rectangles as alternate rectangles for all missing items. You can specify positive or negative integers for these offsets to move an alternate rectangle's coordinates relative to a display rectangle's coordinates.

Use the third element of the missing-items component to supply one of these identifiers: `HMStringItem`, `HMSTRResItem`, `HMStringResItem`, `HMPictItem`, `HMTEResItem`, or `HMSkipItem`, described in "Specifying the Format for Help Messages" on page 3-23. In the remaining four elements of this component, supply the help messages for items in the item list resource that do not otherwise have help messages specified in this `'hdlg'` resource. You can supply either text strings for the help messages or the resource IDs of resources that contain the help messages.

When displaying help balloons for a control, the Help Manager examines the highlight value in the `contrlHilite` field of the control record. An active control that is not selected by the user has a highlight value of 0. Specify a help message for all missing highlighted controls in the fourth element of the missing-items component of the `'hdlg'` resource.

An inactive—that is, dimmed—control has a highlight value of 255. Specify a help message for all missing dimmed controls in the fifth element of the missing-items component.

Note

Don't confuse a disabled item with an inactive control. When you don't want the Control Manager to display visual responses to mouse events in a control, you make it inactive by using the Control Manager procedure `HiliteControl`. When you don't want the Dialog Manager to report events involving an item in a dialog box, you mark it `disabled` in the item list resource. The Dialog Manager makes no visual distinction between disabled and enabled items. See the chapters "Control Manager" and "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for more information. ◆

When, as with checkboxes and radio buttons, the user turns on an on-and-off control, the control has a highlight value of 1. Specify a help message for all missing, active, "on" controls in the sixth element of the missing-items component.

In addition to the values 0, 1, and 255, multipart controls—such as scroll bars—can also take highlight values between 2 and 253, signifying the part code for the part of the control that has been selected by the user. However, you can specify only one message for all possible highlight values that a control might have other than 0, 1, and 255. You can use the seventh element of the missing-items component to specify this message for missing controls.

The following section offers guidelines about what sorts of messages to provide for different types of controls according to their states. For more detailed information about controls, see the chapter "Control Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

When displaying help for items that are not controls, the Help Manager examines only whether the item is enabled or disabled, as specified in the item list resource. For an enabled item (other than a control), you specify a help message in the fourth element of its component in the `'hdlg'` resource. In the fifth element, you specify the help message for the item when it is disabled. The sixth and seventh elements apply only to controls. You should supply these elements with either empty strings or resource IDs of 0, depending on the format indicated by the identifier you specified in the third element of the component.

Specifying Help for Items in an Alert or Dialog Box

After the missing-items component, create **dialog-item components** that specify help messages for the individual items. The first dialog-item component must relate to the item number indexed in the header component; list the remaining dialog-item components in the same order in which they appear in the item list resource. (See the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for information on the item list resource.)

Use the first element of a dialog-item component to specify the coordinates of the help balloon's tip for that item. Use coordinates local to the item's display rectangle (which is specified in the item list resource) to specify the tip. You can specify (0,0) to place the tip 10 pixels from the right and 10 pixels from the bottom of the item's display rectangle.

Use the second element of a dialog-item component to specify an alternate rectangle for the item. Note that you cannot specify hot rectangles—only alternate rectangles—in an `'hdlg'` resource. This is because the Help Manager uses the display rectangles specified in the item list resource as the hot rectangles for help balloons. (If you must specify hot rectangles that are different from the items' rectangles, use the `'hrct'` resource as described in "Specifying Help for Rectangles in Windows" on page 3-67.) You can, however, specify alternate rectangles in `'hdlg'` resources that are different from the display rectangles defined in the item list resource. Alternate rectangles give you additional flexibility in positioning your help balloons onscreen. If you make your alternate rectangle smaller than the display rectangle, for example, you have greater assurance that the Help Manager will be able to fit the help balloon onscreen; if you specify an alternate rectangle that is larger than the display rectangle, you have greater assurance that the help balloon will not obscure some important portion within the display rectangle.

Specify offsets from the item's display rectangle if you want an alternate rectangle that is different from the display rectangle. The Help Manager adds the top, left, bottom, and right offsets that you specify to the coordinates of the item's display rectangle. For example, if you specify (0,0,0,0) as the alternate rectangle's offsets, the Help Manager uses the item's display rectangle as its alternate rectangle. You can specify positive or negative integers for these offsets to move the alternate rectangle's coordinates relative to the display rectangle's coordinates.

Use the third element of a dialog-item component to supply one of these identifiers: `HMStringItem`, `HMSTRResItem`, `HMStringResItem`, `HMPictItem`, `HMTEResItem`, or `HMSkipItem`, described in "Specifying the Format for Help Messages" on page 3-23. Note that in any one dialog-item component in the resource, you can specify only one format for all help messages.

The remaining elements in a dialog-item component specify help messages for the related item. As previously described in "Specifying Missing-Item Information" on page 3-54, the Help Manager uses these elements differently according to whether the item is or is not a control. For elements four through seven in a dialog-item component, supply either text strings for the help messages or the resource IDs of resources that contain the help messages.

You do not have to provide a help message for every state of an item. If you do not provide a help message for a particular state, the Help Manager uses the help message specified in the missing-items component. If the missing-items component does not specify a help message for that state, then the Help Manager does not display a help balloon for that state of that item.

In your help balloons for buttons, use the construction "To [perform action], click this button." For example, the help message for the OK button in a Spell Check dialog box should state something similar to "To check the spelling of this document with the options you've chosen, click this button."

For an unselected radio button or checkbox, use the fourth element of the corresponding dialog-item component to describe what happens when the user selects the button or checkbox. For example, an unselected radio button titled "Left" might have a help balloon that states, "To align all text along the left margin of the document, click this button."

For a selected radio button (that is, one that is "on"), use the sixth element of the corresponding dialog-item component to describe what the selected button does, beginning with a verb. At the end of the message, state that the button is selected. For example, a selected radio button titled "Left" might have a help balloon that states, "Aligns all text along the left margin of the document. This option is selected."

For a selected checkbox (that is, one that is "on"), use the sixth element of the corresponding dialog-item component to describe what the selected checkbox does, then describe how to turn off the option. For example, a selected checkbox titled "On Line" might have a help balloon that states, “Your Macintosh is connected to a remote computer. To disconnect, click this box.”

For a radio button or a checkbox that is dimmed, use the fifth element of the corresponding dialog-item component to describe what it does when it is selected. Use a sentence fragment that begins with a verb. Then explain why the radio button or checkbox is not available. For example, a dimmed radio button titled “Left” might have a help balloon that states, “Aligns all text along the left margin of a document. Not available because no documents are open.”

For an editable text item in a dialog box, use the word “here” in your help message to describe the item. Explain what type of information the user should enter, but don’t describe standard Macintosh editing procedures. For example, an editable text item identified by static text reading “Name” might have a help balloon that states, “Type your name here.” Since an editable text item is typically disabled, you’ll use the fifth element of that item’s component to specify a help balloon.

Since users typically don’t interact with your static text items, you generally shouldn’t provide them with help balloons.

You can use the `HMSkipItem` identifier for an item for which you do not want to provide help. If you specify `HMSkipItem`, the Help Manager does not display help balloons for that item, even if the missing-items component specifies a help message.

In most cases, you should try to describe only the item the balloon is pointing to. It may be tempting to discuss the relationships among items, but this much information can become complex and difficult to read. Remember that the user can point at other items to find out what they are. For example, a button titled “Print” might have a help balloon that states, “To print the document with the options you’ve chosen, click this button.” Do not complicate the message with information like “To print the number of copies of the document that you’ve selected to the left, using the printer named at the top of this dialog box,” and so on.

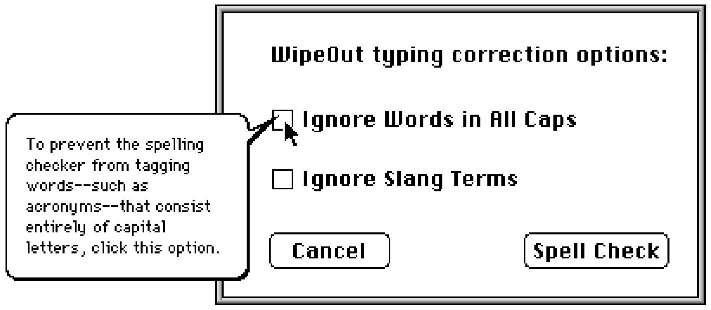
Listing 3-8 shows a sample dialog-item help resource along with its associated item list and string list resources.

Listing 3-8 Rez input for an item list resource and an `'hdlg'` resource

```
resource 'DITL' (145, "Spelling options", purgeable) {
   {  {124, 194, 144, 254},
      Button {
         enabled,
         "OK"
      },
      {48, 23, 67, 202},
      CheckBox {
         enabled,
         "Ignore Words in All Caps"
      },
      {83, 23, 101, 196},
      CheckBox {
         enabled,
         "Ignore Slang Terms"
      },
      {13, 23, 33, 254},
      StaticText {
         disabled,
         "WipeOut typing correction options:"
      },
      /*item for Cancel button goes here*/
      {0,0,0,0},         /*for help balloon: scan 'hdlg' with */
                         /* res ID 145*/
      HelpItem {
         disabled,
         HMScanhdlg      /*scan resource type 'hdlg'*/
         {145}           /*get the resource with ID 145*/
      }
   }
};
resource 'hdlg' (145, "Spell options help", purgeable) {
   /*header component*/
   HelpMgrVersion,       /*version of Help Manager*/
   0,                    /*start help with first item in 'DITL'*/
   hmDefaultOptions,     /*options*/
   0,                    /*balloon definition ID*/
   3,                    /*variation code: hang left of items*/
   /*missing-items component*/
   HMSkipItem {          /*no missing-items help messages*/
      },
   {                     /*help messages for items*/
   /*first dialog-item component: OK button*/
      HMStringResItem { /*store help messages in 'STR#' 145*/
         {10, 10},      /*place tip inside left edge of button*/
         {0,0,0,0},     /*default alternate rectangle: use */
                        /* display rectangle*/
         145, 1,        /*enabled OK button*/
         0, 0,          /*OK button is never dimmed*/
         0, 0,          /*no enabled-and-checked state for */
                        /* button*/
         0, 0           /*no other marked states for button*/
      },
   /*second dialog-item component: All Caps checkbox*/
      HMStringResItem { /*store help messages in 'STR#' 145*/
         {6, 6},        /*place tip in checkbox*/
         {0,0,0,0},     /*default alternate rectangle: use */
                        /* display rectangle*/
         145, 2,        /*highlighted state of checkbox*/
         145, 3,        /*dimmed state of checkbox*/
         145, 4,        /*checkbox is checked*/
         0, 0           /*not applicable to this control*/
      },
   /*third dialog-item component: Slang Terms checkbox*/
      HMStringResItem { /*store help messages in 'STR#' 145*/
         {6, 6},        /*place tip in checkbox*/
         {0,0,0,0},     /*default alternate rectangle: use */
                        /* display rectangle*/
         145, 5,        /*highlighted state of checkbox*/
         145, 6,        /*dimmed state of checkbox*/
         145, 7,        /*checkbox is checked*/
         0, 0           /*not applicable to this control*/
      }
      /*dialog-item component for Cancel button goes here*/
   }
};
resource 'STR#' (145, "Spell options help text") {
   {
   /*[1]*/
   "To check the spelling of this document with the "
      "options you've chosen, click this button.";
   /*[2]*/
   "To prevent the spelling checker from tagging "
      "words--such as acronyms--that consist entirely "
      "of capital letters, click this option.";
   /*[3]*/
   "Prevents the spelling checker from tagging "
      "words--such as acronyms--that consist entirely "
      "of capital letters. Not available until ";
      "you install the main dictionary.";
   /*[4]*/
   "The spelling checker is not tagging "
      "words--such as acronyms--that consist entirely "
      "of capital letters. Click here to make the ";
      "spelling checker tag such words.";
   /*[5]*/
   "To prevent the spelling checker from tagging words "
      "considered to be slang, click this option.";
   /*[6]*/
   "Prevents the spelling checker from tagging "
      "words considered to be slang. "
      "Not available until you install the slang dictionary.";
   /*[7]*/
   "The spelling checker is not tagging "
      "words considered to be slang. "
      "Click here to make the spelling checker tag such words.";
      /*help strings for Cancel button go here*/
   }
};
```

The `'hdlg'` resource in Listing 3-8 specifies help messages for the first three items in the item list resource. Figure 3-17 shows the Help Manager displaying a help balloon for the second item.

Figure 3-17 A help balloon in a modal dialog box


WipeOut typing correction options:
Ignore Words in All Caps
Ignore Slang Terms
Cancel
Spell Check
To prevent the spelling checker from tagging words--such as acronyms--that consist entirely of capital letters, click this option.

Adding a Help Item to an Item List Resource

In Listing 3-8 on page 3-59, an item of type `HelpItem` is included in the item list (`'DITL'`) resource. This item isn't visible to the user; it's provided so that the Help Manager can find the `'hdlg'` or `'hrct'` resource in which you've specified the help messages for your dialog box or alert box.

When creating a help item in an item list resource, specify an empty rectangle—that is, one with coordinates (0,0,0,0)—for the item's display rectangle. Specify `HelpItem` for the item's type, and specify `disabled` for the item's state. Then, specify one of these three identifiers:

Identifier	Purpose
`HMScanhdlg`	For the items in an item list resource, the Help Manager displays the help messages specified in an `'hdlg'` resource.
`HMScanAppendhdlg`	For the items in one item list resource that are appended to those in another item list resource, the Help Manager displays help messages specified in an `'hdlg'` resource.
`HMScanhrct`	For rectangular areas in the dialog box or alert box, the Help Manager displays help messages specified in an `'hrct'` resource.

If you specify help messages for your dialog box or alert box in an `'hdlg'` resource, use either the `HMScanhdlg` or the `HMScanAppendhdlg` identifier. Use the `HMScanAppendhdlg` identifier, however, only when you use the Dialog Manager procedure `AppendDITL` to append the item list resource to another item list resource. The `AppendDITL` procedure is useful, for example, when adding your own items to the standard file dialog box or print dialog box. When you use the `AppendDITL` procedure to add items to an existing dialog box or alert box, the `HMScanAppendhdlg` identifier allows you to provide help balloons for the new items in addition to those balloons already provided for the dialog box or alert box. See the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for more information on the `AppendDITL` procedure.

As described in "Specifying Help for Rectangles in Windows" on page 3-67, you can also use the `'hrct'` resource to specify help balloons for areas of your dialog box or alert box. If you specify help messages for a dialog box or alert box in an `'hrct'` resource, you can use the `HMScanhrct` identifier in the help item of the box's item list resource.

Conclude a help item by specifying the resource ID of the `'hdlg'` or `'hrct'` resource that provides the help messages for the dialog box or alert box.

Using a Help Item Versus Using an 'hwin' Resource

Adding an item of type `HelpItem` to an item list resource is the simplest method of associating the help balloons defined in your `'hdlg'` (or `'hrct'`) resource with the item list resource. A slightly more involved method requires you to create an `'hwin'` (window help) resource. The advantages and disadvantages of the two methods are listed here.

The advantages of adding an item for help to the item list resource are that

- it's simple (you have to create only one resource, the `'hdlg'` or `'hrct'` resource)
- it works for dialog boxes or alert boxes that have no titles and for those whose `windowKind` values do not adequately differentiate them from other windows (the `windowKind` field of window records is described in the chapter "Window Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*)

The disadvantage of adding an item for help to the item list resource is that it allows you to associate help balloons *only* with items listed in the item list resource.

The advantages of using `'hwin'` (window help) resources are that

- you can provide a single help balloon for a group of related items (rather than having separate help balloons for all the items)
- you can provide help balloons for areas instead of items inside the dialog box or alert box

The disadvantages of using `'hwin'` resources are that

- it's slightly more complex, because you must create an `'hwin'` resource in addition to either an `'hdlg'` or an `'hrct'` resource
- it works only for dialog boxes and alert boxes that have titles or `windowKind` values that differentiate them from other windows in your application

Using the `'hwin'` resource requires treating the dialog box or alert box as a static window. When the cursor passes over an active window, the Help Manager attempts to match either the title of the window or the `windowKind` value (from its window record) with a title or `windowKind` value you specify in an `'hwin'` resource. "Associating Help Resources With Static Windows" beginning on page 3-68 describes how to use `'hwin'` resources for dialog boxes, alert boxes, and other kinds of static windows you may wish to define.

Providing Help Balloons for Window Content

You can create help balloons for objects within the content area of your windows. How you choose to provide help balloons for the content area of your windows depends mainly on whether your windows are static or dynamic.

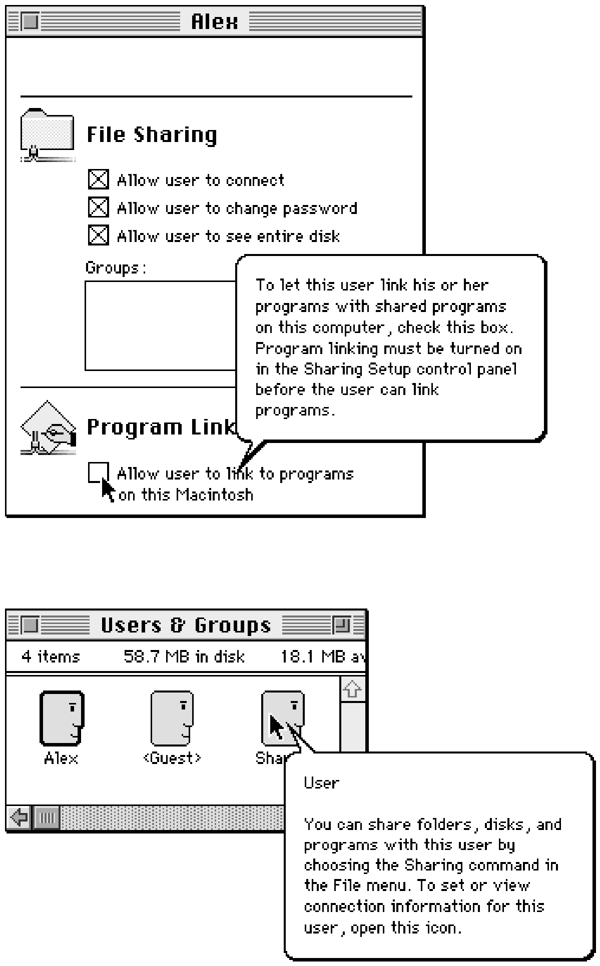
A **static window** doesn't change its title or reposition any of the objects within its content area. A **dynamic window** can reposition any of its objects within the content area, or its title may change.

For example, any window that scrolls past areas of interest to the user is a dynamic window, because the objects with associated help balloons can change location as the user scrolls. A window that displays only a picture that cannot be resized or scrolled is an example of a static window. Figure 3-18 shows examples of static and dynamic windows. “Providing Help Balloons for Static Windows” beginning on page 3-65, “Associating Help Resources With Static Windows” beginning on page 3-68, and “Providing Help Balloons for Dynamic Windows” beginning on page 3-74 describe how to provide help balloons for these types of windows.

Figure 3-18 Static and dynamic windows

Static window

Dynamic window



To let this user link his or her programs with shared programs on this computer, check this box. Program linking must be turned on in the Sharing Setup control panel before the user can link programs.

User

You can share folders, disks, and programs with this user by choosing the Sharing command in the File menu. To set or view connection information for this user, open this icon.

Providing Help Balloons for Static Windows

To provide help balloons for the static windows of your application without modifying its code, create a resource of type `'hwin'` (window help) and additional resources of type `'hrct'` (rectangle help) or `'hdlg'` (dialog-item help). With these resources, the Help Manager automatically tracks the cursor and displays and removes help balloons as the cursor moves into and out of the hot rectangles associated with these resources.

The `'hwin'` resource allows you to associate `'hrct'` and `'hdlg'` resources with your static windows. You use the `'hrct'` and `'hdlg'` resources to define help balloons for the individual objects within your windows. While the Help Manager uses the display rectangles defined in the item list resource as the hot rectangles for `'hdlg'` resources, you can specify your own hot rectangles for alert boxes and dialog boxes and other static windows by using `'hrct'` resources.

Note
BalloonWriter gives nonprogrammers an easy, intuitive way to create help balloons for static windows and dialog and alert boxes. BalloonWriter creates `'hdlg'`, `'hwin'`, and `'hrct'` resources as appropriate and places them in the resource file of your application; BalloonWriter likewise creates and stores `'STR '`, `'STR#'`, and `'TEXT'` resources that contain the help messages authored by nonprogramming writers. ◆

You use an `'hrct'` resource to specify tip coordinates, hot rectangles, balloon definition functions, variation codes, and help messages for areas within a static window.

As explained in “Providing Help Balloons for Items in Dialog Boxes and Alert Boxes” on page 3-51, you use the `'hdlg'` resource to specify the tip, alternate rectangle, and help messages for items in an alert box or dialog box. “Using a Help Item Versus Using an 'hwin' Resource” on page 3-63 describes how to associate either an `'hdlg'` or an `'hrct'` resource with an alert box or a dialog box by adding an item of type `HelpItem` to the box’s item list resource. This section describes how you can instead treat your dialog boxes or alert boxes as static windows and use an `'hwin'` resource instead of `HelpItem` items to associate `'hdlg'` and `'hrct'` resources with the boxes.

The `'hwin'` resource identifies windows by their titles or by their `windowKind` values. You can list all of your windows within one `'hwin'` resource, or you can create separate `'hwin'` resources for your separate windows. (You’ll probably find it easier to maintain your window help if you create only one `'hwin'` resource, but, as described later in this section, you must create separate `'hwin'` resources for windows that require different options. For example, one window may be matched to its `'hwin'` resource by a string *anywhere* in the window’s title, and another window may be matched to its `'hwin'` resource only by the *exact* string of the window’s title.) An `'hwin'` resource contains the resource ID (or IDs) of one or more `'hrct'` or `'hdlg'` resources. With an `'hwin'` resource, you can use both `'hrct'` and `'hdlg'` resources for various parts of the same window.

To use an `'hwin'` resource, the window's window record must specify either a title or a `windowKind` value that adequately distinguishes it from other windows. Within an `'hwin'` resource, you could identify the Verb Tenses window shown in Figure 3-20 on page 3-72 by its title, and you could identify the palette window in Figure 3-19 on page 3-70 by its `windowKind` value.

The chapter "Window Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* describes the `windowKind` field of the window record. Note that `windowKind` values of 0, 1, and 3 through 7 are reserved by system software and that dialog boxes or alert boxes must have a value of 2. Because your dialog boxes and alert boxes must have a `windowKind` value of 2, you can use this value to define only one `'hwin'` resource for all untitled dialog boxes and alert boxes. You may find it difficult—using `'hwin'`, `'hrct'`, and `'hdlg'` resources alone—to provide help balloons for untitled dialog and alert boxes. However, you can use an `'hwin'` resource to associate generic help for the common objects of all your untitled dialog boxes and alert boxes, and you can use the `HMSetDialogResID` function to provide help for the unique objects among them. The `HMSetDialogResID` function is explained on page 3-117.

You describe the tip, a rectangle, and help messages for each object in static windows using either `'hrct'` or `'hdlg'` resources. As shown here, an `'hrct'` resource consists of two types of components: a header component and a hot-rectangle component. You use **hot-rectangle components** to specify the hot rectangles within the window and the help messages for each hot rectangle. (For a description of `'hdlg'` resources, see "Providing Help Balloons for Items in Dialog Boxes and Alert Boxes" beginning on page 3-51.)

Component	**Element**
Header	Help Manager version
	Options
	Balloon definition function
	Variation code
First hot rectangle	Identifier for help message
	Tip's coordinates
	Hot rectangle coordinates
	Help message for hot rectangle
Next hot rectangle	(Same as for first hot rectangle)
	.
	.
	.
Last hot rectangle	(Same as for first hot rectangle)

Specifying Header Information for the 'hrct' Resource

As with the other help resources, specify the `HelpMgrVersion` constant for the first element of the header component of the `'hrct'` resource. For the second element, specify a constant (normally, `hmDefaultOptions`) or the sum of several constants' values from the following list. ("Specifying Options in Help Resources" beginning on page 3-25 describes these options.)

```
CONST hmDefaultOptions     = 0;  {use defaults}
      hmUseSubID           = 1;  {use subrange resource IDs }
                                 { for owned resources}
      hmAbsoluteCoords     = 2;  {ignore coords of window }
                                 { origin and treat upper-left }
                                 { corner of window as 0,0}
      hmSaveBitsNoWindow   = 4;  {don't create window; save }
                                 { bits; no update event}
      hmSaveBitsWindow     = 8;  {save bits behind window }
                                 { and generate update event}
```

Specify the balloon definition function and the variation code (both typically 0) in the third and fourth elements, respectively, of the header component. (The balloon definition function and the variation code are described in detail in "Specifying Header Information for the 'hmnu' Resource" on page 3-32.)

Specifying Help for Rectangles in Windows

Following the header component, use hot-rectangle components to specify tip coordinates, hot rectangles, and help messages for all the areas in the window that would benefit from having help balloons.

For the first element of each hot-rectangle component, specify the format that the help messages take. As with the other help resources, specify the format using one of these identifiers: `HMStringItem`, `HMSTRResItem`, `HMStringResItem`, `HMPictItem`, `HMTEResItem`, or `HMSkipItem`, described in "Specifying the Format for Help Messages" on page 3-23.

Use the second element of the hot-rectangle component to specify the coordinates (local to the window) of the balloon tip. Use the third element to specify the coordinates (local to the window) of the hot rectangle. Use the fourth element to specify your help message, as either a text string or a resource ID.

In a hot-rectangle component, you specify the tip, hot rectangle, and help message for every applicable area in the window. As explained in "Using a Help Item Versus Using an 'hwin' Resource" on page 3-63, you can associate an `'hrct'` resource with an alert box or a dialog box by adding an item of type `HelpItem` to the box's item list resource. For windows, you create an `'hwin'` resource that contains the resource ID of this `'hrct'` resource and that associates the `'hrct'` resource with the window. In either case, the Help Manager automatically tracks the cursor and displays and removes help balloons as the user moves the cursor into and out of the hot rectangles defined in this `'hrct'` resource.

If you need to supply a help balloon for an area within a larger area that needs a different help balloon, create `'hrct'` resources for both the inner and outer areas and specify their areas as hot rectangles. In your resource file, list the `'hrct'` resource for the inner area before the `'hrct'` resource for the outer area. Then, when the cursor is in the inner hot rectangle, the Help Manager scans its `'hrct'` resource first and displays its help balloon instead of the help balloon for the outer hot rectangle. When the cursor moves from the inner hot rectangle to the outer, the Help Manager removes the inner area's help balloon and displays the balloon for the outer hot rectangle.

As previously explained, you can create an `'hdlg'` resource to specify the tips, alternate rectangles, balloon definitions, variation codes, and help messages for items in an item list resource, and you can use an `'hwin'` resource to associate that `'hdlg'` resource with a dialog box or alert box. When help is enabled and your application calls the Dialog Manager routine `ModalDialog`, `IsDialogEvent`, `Alert`, `NoteAlert`, `CautionAlert`, or `StopAlert`, the Help Manager automatically tracks the cursor and displays and removes help balloons for items specified in the `'hdlg'` resource.

Associating Help Resources With Static Windows

To associate `'hrct'` and `'hdlg'` resources with static windows, create an `'hwin'` resource. As shown here, an `'hwin'` resource consists of two types of components: a header component and a window component. Use a **window component** to associate an `'hrct'` or `'hdlg'` resource with a particular window.

Component	Element
Header	Help Manager version
	Options
First window	Resource ID of an `'hrct'` or `'hdlg'` resource
	Resource type (`'hdlg'` or `'hrct'`)
	Length used to compare title strings—or, if flagged by a minus sign (–), the `windowKind` value of an untitled window
	Window title string—or empty string if window is untitled
Next window	(Same as the first window component)
	.
	.
	.
Last window	(Same as the first window component)

Specifying Header Information for the 'hwin' Resource

Specify the HelpMgrVersion constant for the first element of the header component. For the second element, specify a constant (normally, hmDefaultOptions) or the sum of several constants' values from this list.

```
CONST hmDefaultOptions  = 0      {use defaults}
      hmUseSubID        = 1;     {use subrange resource IDs }
                                 { for owned resources}
      hmMatchInTitle    = 16;    {match window by string }
                                 { anywhere in title string}
```

Notice that options regarding local coordinates and bits behind the balloon are not applicable to the 'hwin' resource, but, compared to the other resources related to the Help Manager, the 'hwin' resource has a unique option: hmMatchInTitle.

If you're providing help balloons for a desk accessory or a driver that owns other resources, use the hmUseSubID constant in the second element. (See the chapter "Resource Manager" in this book for a discussion of owned resources and their resource IDs.)

You can specify the hmMatchInTitle constant to match windows containing a specified number of sequential characters starting with any character position in the window title. If you do not specify the hmMatchInTitle constant for the second element of the header component, the Help Manager matches characters starting with the first character of the window title.

For example, if an 'hwin' resource specifies the hmMatchInTitle constant in the header component, specifies in the window component that four characters should be matched, and specifies the character string Test as the window's title string, the Help Manager uses this 'hwin' resource when the cursor is located in any active window that is titled Test, Window Test, or Test Case or is given a title with any other string that contains the characters Test.

If you supply the hmDefaultOptions constant, the Help Manager treats the resource IDs in this resource as regular resource IDs and not as subrange IDs, and it begins matching characters at the first character of the window strings specified in each window component. As long as the window components all use the same options, you can list help for all your windows in a single 'hwin' resource. You must create separate 'hwin' resources for window components that require different options.

Specifying 'hdlg' or 'hrct' Resources in the 'hwin' Resource

You can specify multiple window components after the header component.

Within the 'hwin' resource you identify 'hrct' resources and 'hdlg' resources by their resource IDs and by their types. Use the first element of a window component to specify the resource ID of either an 'hrct' or an 'hdlg' resource. Use the second element to specify that resource's type—either 'hrct' or 'hdlg'. Use the next two elements to specify the window with which you want to associate the 'hrct' or 'hdlg' resource identified in the first two elements.

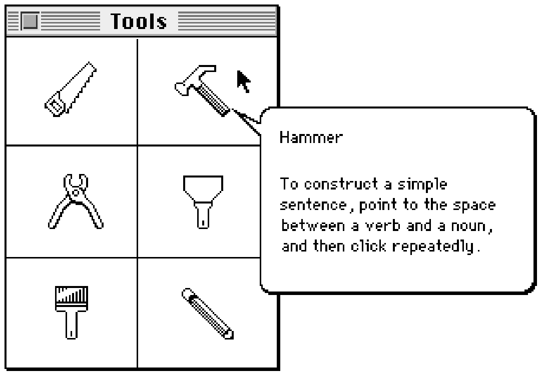
You specify windows in one of these two ways:

- by specifying the number of characters used for matching a window title in the third element of the window component, and by specifying a string consisting of this number of sequential characters from the window's title in the fourth element
- by flagging the third element of the component with a minus sign (–), specifying the `windowKind` value from the window's window record in the third element, and leaving an empty string in the fourth element

When an active window has a title or `windowKind` value that matches an `'hwin'` resource, the Help Manager provides help balloons for the hot rectangles associated with the specified `'hrct'` and `'hdlg'` resources.

Figure 3-19 shows a sample palette an application might use and the help balloon displayed for the hammer tool.

Figure 3-19 A tool palette with a help balloon


Tools
Hammer
To construct a simple sentence, point to the space between a verb and a noun, and then click repeatedly.

Note that the help message in Figure 3-19 names the tool. It's a good idea to name tools, because the name of a tool often helps the user determine the purpose of the tool. After naming the tool, describe one or two likely ways to use it. Don't describe every shortcut or trick you can do with the modifier keys.

For dialog boxes and alert boxes, you can use `'hrct'` resources to define hot rectangles in addition to or instead of those associated with the items. For example, you might want to use an `'hwin'` and an `'hrct'` resource in a dialog box to associate a single help balloon with a group of related items rather than provide separate help balloons for all the individual items. (To provide help balloons for individual items by using `'hdlg'` resources alone, see "Providing Help Balloons for Items in Dialog Boxes and Alert Boxes" beginning on page 3-51.)

When providing one help balloon for a group of options in a dialog box, describe first how to implement the options, and then describe how to tell whether an option is selected. If, for example, radio buttons titled Left, Right, and Middle appear in a dialog box grouped under the heading Alignment, a single help balloon explaining this group might state, "To line up the selected text along the left margin, right margin, or middle of the page, click one of these buttons. A dot indicates the selected option." A help balloon for several checkboxes grouped under the heading Style might state, "To apply design elements to the selected text, click the styles you want. To remove design elements, click the styles you want to remove. An X means a style has been applied."

Listing 3-9 shows the `'hwin'` resource and the `'hrct'` resource for the palette in Figure 3-19.

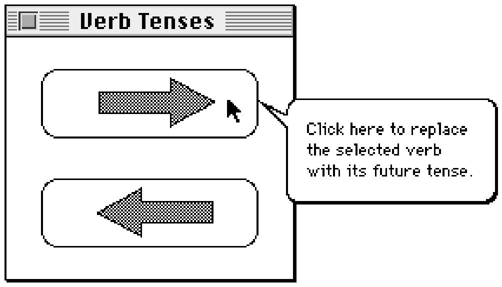
Listing 3-9 Rez input for corresponding `'hwin'` and `'hrct'` resources

```
resource 'hwin' (128, "Window help resource", purgeable) {
   HelpMgrVersion, hmDefaultOptions,/*header component*/
   {                                /*window component*/
      128,        /*resource ID of type specified on next line*/
      'hrct',     /*resource type for defining help*/
      5,          /*length to use when comparing strings*/
      "Tools"     /*window's title string*/
   }
};
resource 'hrct' (128, "Tools palette help") {
   /*header component*/
   HelpMgrVersion,
   hmDefaultOptions,
   0,                /*balloon definition function*/
   0,                /*variation code*/
   {
   /*hot-rectangle component for saw tool goes here*/
   /*hot-rectangle component for hammer tool*/
   HMStringResItem {
      {50, 127},        /*tip's coordinates*/
      {22,99,54,131},   /*hot rectangle*/
      147,2             /*'STR#' resource ID and index*/
      }
   /*hot-rectangle components for other tools go here*/
   }
};

resource 'STR#' (147, "Tools palette help text") {
   {
   /*[1] saw tool*/
   /*help text for saw tool goes here*/
   /*[2] hammer tool*/
   "Hammer\n\nTo construct a simple sentence, point to the "
      "space between a verb and a noun, and then click "
      "repeatedly.";
   /*help for other tools goes here*/
   }
};
```

You can also use the 'hwin' resource to associate help for items in an alert box or a dialog box. Figure 3-20 shows the Help Manager displaying a help balloon for an item in the dialog box titled Verb Tenses.

Figure 3-20 A help balloon for a dialog box with a title


Verb Tenses
Click here to replace the selected verb with its future tense.

Listing 3-10 shows how the 'hwin' resource associates an 'hdlg' resource with the dialog box illustrated in Figure 3-20. This 'hwin' resource associates help with three different windows: the first is the window titled Tools, the second is an untitled window with a windowKind value of 10, and the third is the dialog box titled Verb Tenses.

Listing 3-10 Rez input for specifying help for titled and untitled windows

```
resource 'hwin' (128, "Window help resource", purgeable) {
   /*header component*/
   HelpMgrVersion, hmDefaultOptions,
      {  /*first window component*/
      128,              /*help resource ID for Tools window*/
      'hrct',           /*resource type for defining help*/
      5,                /*length to use when comparing strings*/
      "Tools",          /*window's title string*/
      /*second window component*/
      129,              /*help resource ID for untitled window*/
      'hdlg',           /*resource type for defining help*/
      -10,              /*match on windowKind values of 10*/
      "",               /*matching on windowKind, so empty */
                        /* string goes here*/
      /*third window component*/
      130,              /*help res ID for Verb Tenses window*/
      'hdlg',           /*resource type for defining help*/
      11,               /*length to use when comparing strings*/
      "Verb Tenses",    /*dialog box's title string*/
   }
};
resource 'hdlg' (130, "Help for Verb Tense control", purgeable) {
   /*header component*/
   HelpMgrVersion,      /*version of Help Manager*/
   0,                   /*start with first item in item list*/
   hmDefaultOptions,    /*options*/
   0,                   /*balloon definition ID*/
   0,                   /*variation code*/
   /*missing-items component*/
   HMSkipItem  {/*no missing-item help message*/
      },
   {  /*first dialog-item component*/
      HMStringResItem {
         {20, 130},     /*tip--local to item's display rectangle*/
         {0,0,0,0},     /*default alternate rectangle: use */
                        /* item's display rectangle*/
         131, 1,        /*highlighted control for future tense*/
         131, 2,        /*dimmed control for future tense*/
         0, 0,          /*no checked state for control*/
         0, 0           /*no other states for control*/
      },
      /*second dialog-item component*/
      HMStringResItem {
         {20, 130},     /*tip--local to item's display rectangle*/
         {0,0,0,0},     /*default alternate rectangle: use */
                        /* item's display rectangle*/
         131, 3,        /*highlighted control for past tense*/
         131, 4,        /*dimmed control for past tense*/
         0, 0,          /*no enabled-and-checked control*/
         0, 0           /*no other marks for control*/
      }
   }
};
resource 'STR#' (131, "Verb tense help strings") {
   {
   /*[1] highlighted control for future tense: help text*/
   "Click here to replace the selected verb with its "
      "future tense.";
   /*[2] dimmed control for future tense: help text*/
   "Click here to replace a verb with its future tense. "
      "Not available now because you have not selected a verb.";
   /*[3] /*highlighted control for past tense: help text*/
   "Click here to replace the selected verb with its past tense.";
   /*[4] dimmed control for past tense: help text*/
   "Click here to replace a verb with its past tense. "
      "Not available now because you have not selected a verb.";
   }
};
```

Providing Help Balloons for Dynamic Windows

To create help balloons for objects whose location in the content area of windows may vary, your application needs to use Help Manager routines to display and remove balloons as the user moves the cursor.

Note

Nonprogrammers can use the BalloonWriter tool to provide you with delimited ASCII text that you can then use in conjunction with Help Manager routines to display balloons for dynamic windows. However, BalloonWriter does not create the resources or routines necessary to automatically display help balloons for these types of windows. ◆

You should display or remove help balloons for dynamic windows at the same time that you normally check the mouse location to display or change the cursor. For example, if you provide your own `DoIdle` procedure (as described in the chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*), you can also check the mouse location and, if the cursor is located in a hot rectangle, you should display the associated help balloon.

To create help balloons for the content area of a dynamic window, you need to

- identify the hot rectangles for each area or object
- create data structures to store the locations of the hot rectangles
- determine how to calculate their changing locations
- track and update the hot rectangles
- use the `HMShowBalloon` function to display a help balloon when the cursor is located in a hot rectangle

After defining all the hot rectangles within your content region, create separate `'STR '`, `'STR#'`, `'PICT'`, or `'TEXT'` and `'styl'` resources for the help balloons' messages. You don't have to store the help messages in these resources when using `HMShowBalloon`, but doing so makes your application easier to localize.

When you use the HMShowBalloon function, your application is responsible for tracking the cursor and determining when to display the help balloon. If you use the HMShowBalloon function, you can let the Help Manager track the cursor and determine when to remove the help balloon, or your application can remove the balloon when necessary by calling the HMRemoveBalloon function. If you display your own help balloons using the HMShowBalloon function, you should use the HMGetBalloons function to determine whether help is enabled before displaying a help balloon. If help is not enabled, you don't need to call any Help Manager routines that display balloons, because they won't do anything unless HMGetBalloons returns TRUE.

The HMShowBalloon function is useful for

- windows whose content changes
- windows that can be resized
- windows that contain hot rectangles with variable locations
- situations in which you want your application to have more control over the display and removal of the help balloon

For example, windows with scrolling icons (such as the Users & Groups dynamic window shown in Figure 3-18 on page 3-64) require you to use HMShowBalloon to display help balloons for the icons. Likewise, if you have tools—such as rulers that users configure for tab stops in a word-processing document—that scroll with a document, you'll need to use HMShowBalloon to display help balloons for the scrolling tools.

When using HMShowBalloon, you specify the help message, the balloon tip's coordinates, an alternate rectangle to use if the Help Manager needs to move the tip, an optional pointer to a function that can modify the tip and alternate rectangle coordinates, the balloon definition function, and the variation code. In the final parameter to the HMShowBalloon function, provide a constant that tells the Help Manager whether to save the bits behind the balloon.

```
myErr := HMShowBalloon(aHelpMsg, tip, alternateRect, tipProc,
                       theProc, variant, method);
```

Specify the help message in a help message record, which you pass in the aHelpMsg parameter to the HMShowBalloon function. You can specify the help message for each hot rectangle using text strings, 'STR ' resources, 'STR#' resources, styled text resources, 'PICT' resources, handles to styled text records, or handles to pictures.

The HMMessageRecord data type defines the help message record.

```
TYPE HMMessageRecord =
RECORD
   hmmHelpType:         Integer;             {type of next field}
   CASE Integer OF
      khmmString:       (hmmString: Str255); {Pascal string}
      khmmPict:         (hmmPict: Integer);  {'PICT' resource ID}
      khmmStringRes:    (hmmStringRes: HMStringResType);
                                             {'STR#' resource }
                                             { ID and index}
      khmmTEHandle:     (hmmTEHandle: TEHandle);
                                             {TextEdit handle}
      khmmPictHandle:   (hmmPictHandle: PicHandle);
                                             {picture handle}
      khmmTERes:        (hmmTERes: Integer); {'TEXT'/'styl' }
                                             { resource ID}
      khmmSTRRes:       (hmmSTRRes: Integer);{'STR ' resource ID}
END;
```

The hmmHelpType field specifies the data type of the second field of the help message record. You specify one of these constants for the hmmHelpType field.

```
CONST khmmString        = 1;     {Pascal string}
      khmmPict          = 2;     {'PICT' resource ID}
      khmmStringRes     = 3;     {'STR#' resource ID and index}
      khmmTEHandle      = 4;     {TextEdit handle}
      khmmPictHandle    = 5;     {picture handle}
      khmmTERes         = 6;     {'TEXT' and 'styl' resource ID}
      khmmSTRRes        = 7;     {'STR ' resource ID}
```

You specify the help message itself in the second field of the help message record.

You can specify the help message by using a text string, a text string stored in a resource of type 'STR ', or a text string stored as an 'STR#' resource. You can also provide the information using styled text resources, or you can provide a handle to a styled text record. If you want to provide a picture for the help message, you can use a resource of type 'PICT' or provide a handle to a picture.

Listing 3-11 illustrates how to specify a Pascal string using the khmmString constant in the help message record. (Although you can specify a string from within your code, storing the strings in resources and then accessing them through the Resource Manager makes localization easier.)

Listing 3-11 Using a string resource as the help message for `HMShowBalloon`

```
PROCEDURE DoTextStringBalloon;
VAR
   aHelpMsg:         HMMessageRecord;
   tip:              Point;
   alternateRect:    RectPtr;
   err:              OSErr;
BEGIN
   aHelpMsg.hmmHelpType := khmmString;
   aHelpMsg.hmmString := 'To turn the page, click here.';
   MySetTipAndAltRect(tip, alternateRect); {initialize values}
   err := HMShowBalloon(aHelpMsg, tip, alternateRect,
                        NIL, 0, 0, kHMRegularWindow);
END;
```

To use a picture, you can either store the picture as a `'PICT'` resource or create the `'PICT'` graphic from within your application and provide a handle to it. Because the Help Manager uses the resource itself or the actual handle that you pass to `HMShowBalloon`, your `'PICT'` resource should be purgeable, or, when using a handle to a `'PICT'` resource, you should release the handle or dispose of it when you are finished with it.

Listing 3-12 illustrates how to use the `khmmPict` constant for specifying a `'PICT'` resource ID in a help message record. The help message record is then passed in the `aHelpMsg` parameter of the `HMShowBalloon` function.

Listing 3-12 Using a picture resource as the help message for `HMShowBalloon`

```
PROCEDURE DoPictBalloon;
VAR
   aHelpMsg:         HMMessageRecord;
   tip:              Point;
   alternateRect:    RectPtr;
   err:              OSErr;
BEGIN
   aHelpMsg.hmmHelpType := khmmPict;
   aHelpMsg.hmmPict := 128;      {resource ID of 'PICT' resource}
   MySetTipAndAltRect(tip, alternateRect); {initialize values}
   err := HMShowBalloon(aHelpMsg, tip, alternateRect,
                        NIL, 0, 0, kHMRegularWindow);
END;
```

Listing 3-13 illustrates how to specify a handle to a 'PICT' resource using the khmmPictHandle constant in the help message record. The help message record is then passed to the HMShowBalloon function in the aHelpMsg parameter.

Listing 3-13 Using a handle to a picture resource as the help message for HMShowBalloon

```
PROCEDURE DoPictBalloon2;
VAR
   pict:          PicHandle;
   aHelpMsg:      HMMessageRecord;
   tip:           Point;
   pictFrame:     Rect;
   alternateRect: RectPtr;
   err:           OSErr;
BEGIN
   MySetPictFrame(pictFrame); {initialize pictFrame}
   pict := OpenPicture(pictFrame);
   DrawString('Test Balloon');
   ClosePicture;
   aHelpMsg.hmmHelpType := khmmPictHandle;
   aHelpMsg.hmmPictHandle := pict;
   MySetTipAndAltRect(tip, alternateRect); {initialize values}
   err := HMShowBalloon(aHelpMsg, tip, alternateRect,
                        NIL, 0, 0, kHMRegularWindow);
   KillPicture(pict);
END;
```

To specify a help message stored in a string list ('STR#' resource) in a help message record, you must first create a Help Manager string list record. The HMStringResType data type defines a Help Manager string list record.

```
TYPE HMStringResType =
RECORD
   hmmResID:   Integer;    {resource ID of 'STR#' resource}
   hmmIndex:   Integer;    {index of string}
END;
```

The hmmResID field specifies the resource ID of the 'STR#' resource, and the hmmIndex field specifies the index of a string within that resource.

To use a string stored in an 'STR#' resource with the HMShowBalloon function, use the khmmStringRes constant in the hmmHelpType field of the help message record, and supply the hmmStringRes field with a Help Manager string list record, as shown in Listing 3-14.

Listing 3-14 Using a string list resource as the help message for HMShowBalloon

```
PROCEDURE DoStringListBalloon;
VAR
   aHelpMsg:         HMMessageRecord;
   tip:              Point;
   alternateRect:    RectPtr;
   khmmStringRes:    HMStringResType;
   err:              OSErr;
BEGIN
   aHelpMsg.hmmHelpType := khmmStringRes;
   aHelpMsg.hmmStringRes.hmmResID := 1000;
   aHelpMsg.hmmStringRes.hmmIndex := 1;
   MySetTipAndAltRect(tip, alternateRect); {initialize values}
   err := HMShowBalloon(aHelpMsg, tip, alternateRect,
                        NIL, 0, 0, kHMRegularWindow);
END;
```

To use styled text resources with the HMShowBalloon function, use the khmmTERes constant in the hmmHelpType field of the help message record. In the next field, supply a resource ID that is common to both a 'TEXT' resource and a style scrap ('styl') resource. For example, you might create a 'TEXT' resource that contains the words "Displays your text in boldface print." You would also create a 'styl' resource (with the same resource ID as the 'TEXT' resource) that applies boldface style to the word "boldface." When you specify the HMTEResItem constant and the resource ID number of the 'TEXT' and 'styl' resources, the Help Manager employs TextEdit routines to display your text with your prescribed styles.

To use a handle to a styled text record, supply the `khmmTEHandle` constant in the `hmmHelpType` field, as illustrated in Listing 3-15.

Listing 3-15 Using styled text resources as the help message for `HMShowBalloon`

```
PROCEDURE DoStyledTextBalloon;
VAR
   aHelpMsg:         HMMessageRecord;
   tip:              Point;
   alternateRect:    RectPtr;
   hTE:              TEHandle;
   err:              OSErr;
BEGIN
   hTE := TEStyleNew(destRect, viewRect);    {or, use TENew}
   {be sure to fill in data in handle here}
   aHelpMsg.hmmHelpType := khmmTEHandle;
   aHelpMsg.hmmTEHandle := hTE;
   MySetTipAndAltRect(tip, alternateRect); {initialize values}
   err := HMShowBalloon(aHelpMsg, tip, alternateRect,
                        NIL, 0, 0, kHMRegularWindow);
END;
```

When using the `HMShowBalloon` function, you specify the tip in the `tip` parameter and the rectangle pointed to in the `alternateRect` parameter in global coordinates. The Help Manager calculates the location and size of the help balloon. If the help balloon fits onscreen, the Help Manager displays the help balloon using the specified tip.

If you use the previously described help resources to define help balloons, the Help Manager uses the hot rectangles you specify in the help resources for two purposes: first, to associate areas of the screen with help balloons and, second, to move the tip if the help balloon doesn't fit onscreen.

However, if you use the `HMShowBalloon` function to display help balloons, you must identify hot rectangles, create your own data structures to store their locations, track the cursor yourself, and call `HMShowBalloon` when the cursor moves to your hot rectangles. The Help Manager does not know the locations of your hot rectangles, so it cannot use them for moving the tip if the help balloon is placed offscreen. Instead, the Help Manager uses the alternate rectangle that you point to with the `alternateRect` parameter. Often, you specify the same coordinates for the alternate rectangle that you specify for your hot rectangle. However, you may choose to make your alternate rectangle smaller or larger than your hot rectangle. If you make your alternate rectangle smaller than your hot rectangle, you have greater assurance that the Help Manager will be able to fit the help balloon onscreen; if you specify an alternate rectangle that is larger than your hot rectangle, you have greater assurance that the help balloon will not obscure some object explained by the balloon.

By specifying a rectangle in the `alternateRect` parameter, you tell the Help Manager to call `HMRemoveBalloon` to automatically remove the balloon when the cursor leaves the area bounded by the rectangle. However, if you specify `NIL` for the `alternateRect` parameter, your application is responsible for tracking the cursor and determining when to remove the help balloon. When you specify `NIL`, the Help Manager also does not attempt to calculate a new tip position if the help balloon is offscreen.

Use the `tipProc` parameter (the fourth parameter to `HMShowBalloon`) to specify the tip function called by the Help Manager before displaying the balloon. Specify `NIL` to use the Help Manager's default tip function, or supply your own tip function and point to it in this parameter. Writing your own tip function is described in "Application-Defined Routines" beginning on page 3-128.

Note

When you call the `HMShowBalloon` function, the Help Manager does not display the help balloon or attempt to move the tip under either of these conditions:

The help balloon's tip is offscreen or in the menu bar, and you don't specify an alternate rectangle.

Both the help balloon's tip and the alternate rectangle are offscreen. ◆

Use the parameter `theProc` (the fifth parameter) to specify a balloon definition function. To use the standard balloon definition function, specify 0 in this parameter. To use your own balloon definition function, specify the resource ID of the `'WDEF'` resource containing your balloon definition function. Writing your own balloon definition function is described in "Writing Your Own Balloon Definition Function" on page 3-93.

Supply a variation code for the balloon definition function in the `variant` parameter (the sixth parameter to `HMShowBalloon`). Specify 0 in the `variant` parameter to use the default help balloon shape, specify a code from 1 to 7 to use one of the other positions provided by the standard balloon definition function, or specify a code to use one of the positions provided by your own balloon definition function.

Use the `method` parameter (the last parameter to `HMShowBalloon`) to specify how the Help Manager should draw and remove the balloon. Use the following constants for the parameter.

```
CONST kHMRegularWindow     = 0;  {don't save bits; just update}
      kHMSaveBitsNoWindow  = 1;  {save bits; don't do update}
      kHMSaveBitsWindow    = 2;  {save bits; do update event}
```

If you specify `kHMRegularWindow`, the Help Manager draws and removes the help balloon as if it were a window. That is, when drawing the balloon, the Help Manager does not save bits behind the balloon, and when removing the balloon, the Help Manager generates an update event. This is the standard behavior of help balloons; it is the behavior you should normally use.

If you specify `kHMSaveBitsNoWindow` in the `method` parameter, the Help Manager does not create a window for displaying the balloon. Instead, the Help Manager creates a help balloon that is more like a menu than a window. The Help Manager saves the bits behind the balloon when it creates the balloon. When it removes the balloon, the Help Manager restores the bits without generating an update event. You should use this technique only in a modal environment where the bits behind the balloon cannot change from the time the balloon is drawn to the time it is removed. For example, you might specify the `kHMSaveBitsNoWindow` constant when providing help balloons for pop-up menus that overlay complex graphics, which might take a long time to redraw with an update event.

If you specify `kHMSaveBitsWindow`, the Help Manager treats the help balloon as a hybrid having properties of both a menu and a window. That is, the Help Manager saves the bits behind the balloon when it creates the balloon, and when it removes the balloon, it both restores the bits and generates an update event. You'll rarely need this option. It is necessary only in a modal environment that might immediately change to a nonmodal environment—that is, where the bits behind the help balloon are static when the balloon is drawn, but can possibly change before the help balloon is removed.

Listing 3-16 shows a sample routine that displays help balloons for hot rectangles within the content area of a window.

Listing 3-16 Using `HMShowBalloon` to display help balloons

```
PROCEDURE FindAndShowBalloon (window: WindowPtr);
VAR
   i:          Integer;
   mouse:      Point;
   savePort:   GrafPtr;
   helpMsg:    HMMessageRecord;
   inRect:     Boolean;
   hotRect:    Rect;
   result:     OSErr;
BEGIN
   IF (window = FrontWindow) THEN      {only do frontmost windows}
   BEGIN
      GetPort(savePort);   {save the old port for later}
      SetPort(window);     {set the port to the front window}
      GetMouse(mouse);     {get the mouse location in local }
                           { coords}
      inRect := FALSE;     {clear flag saying mouse location }
                           { wasn't in any hot rectangle}
      IF PtInRect(mouse, window^.portRect) THEN
         {if the cursor is in the window}
         FOR i := 1 TO 10 DO  {check all ten predefined hot }
                              { rectangles in the window}
            IF PtInRect(mouse, MyPredefinedRects[i]) THEN
            BEGIN             {the cursor is in a hot rectangle}
               IF (i <> gLastBalloon) THEN
               {user moved cursor to a different hot rectangle}
               BEGIN
                  hotRect := MyPredefinedRects[i];
                  LocalToGlobal(hotRect.topLeft);
                  {converting rect to global}
                  LocalToGlobal(hotRect.botRight);
                  WITH hotRect DO   {put the tip in the middle}
                     SetPt(mouse, (right + left) div 2,
                           (bottom + top) div 2);
                  helpMsg.hmmHelpType := khmmStringRes;
                  {get help message from an 'STR#' resource}
                  helpMsg.hmmStringRes.hmmResID := kHelpMsgsID;
                  helpMsg.hmmStringRes.hmmIndex := i;
                  result := HMShowBalloon
                              (helpMsg,   {use just-made help msg}
                               mouse,     {pointing to this tip}
                               @MyPredefinedRects[i], {use hot }
                                          { rect for alt rect}
                               NIL,       {no special tip proc}
                               0,0,       {using default balloon}
                        kHMRegularWindow);{don't save bits behind}
                  IF (result = noErr) THEN {then remember balloon}
                     gLastBalloon := i;
               END;
               inRect := TRUE;   {remember when the }
                                 { cursor is in any hot rect}
            END;
      IF not inRect THEN
         gLastBalloon := -1;     {clear last balloon global for }
                                 { no hit}
      SetPort(savePort);         {restore the port}
   END;
END;     {FindAndShowBalloon}
```

The `FindAndShowBalloon` procedure in Listing 3-16 tracks the cursor, and, if the cursor is located in a predefined hot rectangle, it displays a help balloon for that rectangle. In this example there are ten predefined rectangles (in the `MyPredefinedRects` array) and ten corresponding help messages in an `'STR#'`

resource (of ID `kHelpMsgsID`)—one message for each hot rectangle. Other supporting routines can update the coordinates of the hot rectangles as their locations change.

You can also use the `HMShowBalloon` function from the event filter function of a modal dialog box or an alert box. See the chapter "Dialog Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for information on event filter functions.

Overriding Help Balloons for Non-Document Icons

The Finder displays default help balloons for all icon types. By specifying an `'hfdr'` resource in your application's resource fork, you can provide your own help balloon for the Finder to display when the user moves the cursor over your non-document icons.

Note
BalloonWriter, available from APDA, is a tool that gives nonprogrammers an easy way to create help balloons for most of the icons that the Finder displays for your software. BalloonWriter creates an `'hfdr'` resource and places it in the resource fork of the file represented by the icon; BalloonWriter likewise creates and stores an `'STR '` resource that contains the help message. ◆

To override the Finder's default help balloons for your application icon, desk accessory icon, system extension icon, or control panel icon, create an `'hfdr'` resource in your resource file. As shown here, an `'hfdr'` resource consists of two components: a header component and an icon component. Use the **icon component** to specify a help message for your application's Finder icon.

Component	Element
Header	Help Manager version
	Options
	Balloon definition function
	Variation code
Icon	Identifier for help message
	Help message for application icon

Note
You cannot override the default help balloon that the Finder uses for document icons. ◆

Use resource ID –5696 for your `'hfdr'` resource. If an `'hfdr'` resource with that ID exists for an application, the Help Manager uses it instead of the default help balloon supplied by the Finder.

Specifying Header Information for the 'hfdr' Resource

As with the other help resources, specify the `HelpMgrVersion` constant for the first element of the header component of the `'hfdr'` resource. For the second element, specify a constant (normally, `hmDefaultOptions`) or the sum of several constants' values from the following list. ("Specifying Options in Help Resources" beginning on page 3-25 describes these options.)

```
CONST hmDefaultOptions     = 0;  {use defaults}
      hmUseSubID           = 1;  {use subrange resource IDs }
                                 { for owned resources}
      hmAbsoluteCoords     = 2;  {ignore coords of window }
                                 { origin and treat upper-left }
                                 { corner of window as 0,0}
      hmSaveBitsNoWindow   = 4;  {don't create window; save }
                                 { bits; no update event}
      hmSaveBitsWindow     = 8;  {save bits behind window }
                                 { and generate update event}
```

Specify the balloon definition function and variation code (both typically 0) in the third and fourth elements, respectively, of the header component. (These are described in detail earlier in "Specifying Header Information for the 'hmnu' Resource" on page 3-32.)

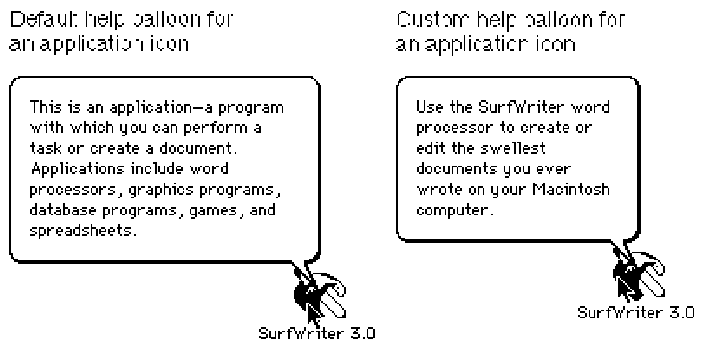
Specifying Help for an Icon

In the icon component, use the first element to specify the format that the help message takes. As with the other help resources, specify the format using one of these identifiers: `HMStringItem`, `HMSTRResItem`, `HMStringResItem`, `HMPictItem`, `HMTEResItem`, or `HMSkipItem`. These identifiers are described in "Specifying the Format for Help Messages" on page 3-23. (If you specify `HMSkipItem`, no help balloon appears.)

In the second element of the icon component, specify the help message. Your help message doesn't have to describe how to open icons; you can assume that users know how.

Figure 3-21 shows the default help balloon for application icons on the left. A custom help balloon for the same icon appears on the right.

Figure 3-21 Default and custom help balloons for an application icon


Default help balloon for an application icon
This is an application—a program with which you can perform a task or create a document. Applications include word processors, graphics programs, database programs, games, and spreadsheets.
SurfWriter 3.0
Custom help balloon for an application icon
Use the SurfWriter word processor to create or edit the swellest documents you ever wrote on your Macintosh computer.
SurfWriter 3.0

The custom help balloon on the right side of Figure 3-21 is supplied with the resources shown in Listing 3-17.

Listing 3-17 Rez input for creating an `'hfdr'` resource for an application icon

```
resource 'hfdr' (-5696) {     /*help for SurfWriter icon*/
      /*header component*/
   HelpMgrVersion, hmDefaultOptions, 0, 0,
   {  /*icon component*/
   HMSTRResItem {             /*use 'STR ' resource 1001*/
      1001
      }
   }
};
resource 'STR ' (1001) {      /*help message for SurfWriter icon*/
   "Use the SurfWriter word processor to wrote or edit the "
   "swellest documents you ever wrote on "
   "your Macintosh computer."
};
```

Overriding Other Default Help Balloons

The Help Manager also provides default help balloons for the title bar and the close and zoom boxes of an active window, for the windows of inactive applications, for inactive windows of an active application, and for the area outside a modal dialog box.

Apple Computer, Inc., has researched and tested these help messages to ensure that they are as effective as possible for users. Normally, you shouldn't need to override them. However, you can override one or more of these defaults if you feel you absolutely must by creating a resource of type `'hovr'`.

Using an `'hovr'` resource sets the default help balloons for your application only. It does not affect the default help balloons used by other applications.

An `'hovr'` resource consists of exactly nine components: a header component, a missing-items component, and seven components that specify help messages for seven standard user interface features.

Component	**Element**
Header	Help Manager version
	Options
	Balloon definition function
	Variation code
Missing-items help	Identifier for help message
	Help message for items missing from this resource or lacking help messages
Title bar help	Identifier for help message
	Help message for title bar of active window
Reserved	`HMSkipItem` identifier (always used here)
	No help message; reserved for future use
Close box help	Identifier for help message
	Help message for close box of active window
Zoom box help	Identifier for help message
	Help message for zoom box of active window
Help for active application's inactive windows	Identifier for help message
	Help message for inactive window of active application
Help for inactive application's windows	Identifier for help message
	Help message for window of inactive application
Help for area outside a modal dialog box or alert box	Identifier for help message
	Help message for area outside a modal dialog box or an alert box

Specifying Header Information for the 'hovr' Resource

As with the other help resources, specify the `HelpMgrVersion` constant for the first element of the header component of the `'hovr'` resource. For the second element, specify a constant (normally, `hmDefaultOptions`) or the sum of several constants' values from the following list. ("Specifying Options in Help Resources" beginning on page 3-25 describes these options.)

```
CONST hmDefaultOptions     = 0;  {use defaults}
      hmUseSubID           = 1;  {use subrange resource IDs }
                                 { for owned resources}
      hmAbsoluteCoords     = 2;  {ignore coords of window }
                                 { origin and treat upper-left }
                                 { corner of window as 0,0}
      hmSaveBitsNoWindow   = 4;  {don't create window; save }
                                 { bits; no update event}
      hmSaveBitsWindow     = 8;  {save bits behind window }
                                 { and generate update event}
```

Specify the balloon definition function and variation code (both typically 0) in the third and fourth elements, respectively, of the header component. (The balloon definition function and variation code are described in detail earlier in "Specifying Header Information for the 'hmnu' Resource" on page 3-32.)

Overriding Default Help

In the first element of the missing-items component, supply an identifier. As with the other help resources, use one of these identifiers: `HMStringItem`, `HMSTRResItem`, `HMStringResItem`, `HMPictItem`, `HMTEResItem`, or `HMSkipItem`. These identifiers are described in "Specifying the Format for Help Messages" on page 3-23. For the second element, supply either a text string for the help message or the resource ID of the resource that contains the help message.

The Help Manager expects the remaining components of an `'hovr'` resource to be listed in the order previously shown. If you specify fewer than seven components in the Rez input file, the Help Manager adds components to the end of your list until there are seven. Each component that the Help Manager adds uses the message specified in the missing-items component. The Help Manager also uses the missing-items component's help message if the Rez input file specifies an empty string or a resource ID of 0 for any other component's help message.

For the first element of each of the remaining components, specify one of these identifiers: `HMStringItem`, `HMSTRResItem`, `HMStringResItem`, `HMPictItem`, `HMTEResItem`, or `HMSkipItem`. To use any of the default help balloons, use the `HMSkipItem` identifier. For the second element of each of the remaining components, supply either a text string for the help message or the resource ID of the resource that contains the help message.

Listing 3-18 shows a resource of type `'hovr'` that overrides all of the default help balloons.

Listing 3-18 Rez input for an `'hovr'` resource

```
resource 'hovr' (1000) {
   /*header component*/
   HelpMgrVersion,
   hmDefaultOptions, /*options*/
   0,       /*the balloon definition ID*/
   0,       /*variation code*/
   /*missing-items component*/
   HMStringItem {    /*missing items in case this resource is */
                     /* short of components*/
      "Missing override message"
      },
   {
   /*remaining components: for overriding default messages*/
      HMSkipItem {         /*title bar help*/
         /*HMSkipItem means use default help balloon for this element*/
      },
      HMSkipItem {      /*reserved; always specify HMSkipItem*/
      },
      HMStringItem {    /*close box help*/
         ""             /*empty string means use missing-items help*/
      },
      HMStringItem {    /*zoom box help*/
         "Get this message if in Zoom In or Zoom Out box."
      },
      HMStringItem {    /*help for active app's inactive window*/
         "Get this message if in inactive window of "
         "active application."
      },
      HMStringItem {    /*help for inactive app's window*/
         "Get this message if in window of inactive application."
      },
      HMStringItem {    /*help when outside of modal dialog box*/
         "Get this message if outside modal dialog box."
      }
   }
};
```

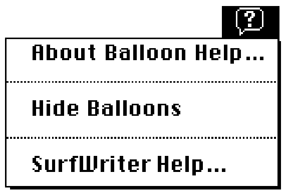
Adding Menu Items to the Help Menu

The Help menu is specific to each application, just as the File and Edit menus are. The Help menu items defined by the Help Manager should be common to all applications, but you can add your own menu items for help-related information.

If you provide your users with help information in addition to help balloons, you should append a command to the Help menu for this information. The Help menu gives users one consistent place to obtain help information.

When adding your own items to the Help menu, include the name of your application in the command so that users can easily determine which application the help command relates to. For example, Figure 3-22 shows the Help menu with an item appended to it by the active application.

Figure 3-22 The Help menu with an appended menu item



You add items to the Help menu by using the `HMGetHelpMenuHandle` function and by providing an `'hmnu'` resource and specifying the `kHMHelpMenuID` constant as the resource ID.

The `HMGetHelpMenuHandle` function returns a copy of the handle to the Help menu. Do not use the Menu Manager function `GetMenuHandle` to get a handle to the Help menu, because `GetMenuHandle` returns a handle to the global Help menu, not the Help menu that is specific to your application. Once you have a handle to the Help menu that is specific to your application, you can add items to it using the `AppendMenu` procedure or other Menu Manager routines. For example, this code adds the menu item displayed in Figure 3-22.

```
VAR
   mh:   MenuHandle;
   err:  OSErr;

BEGIN
   err := HMGetHelpMenuHandle(mh);
   IF err = noErr THEN
      IF mh <> NIL THEN
         BEGIN
            AppendMenu(mh, 'SurfWriter Help…');
         END;
   DrawMenuBar;
END;
```

Be sure to use an 'hmnu' resource to provide help balloons for items you've added to the Help menu. Use the kHMHelpMenuID constant (–16490) to specify the 'hmnu' resource ID. After the header component of the 'hmnu' resource, provide a missing-items component and then the components for your appended items. You don't provide a menu-title component here; instead, the Help Manager automatically creates the help balloons for the Help menu title and the standard Help menu items. The Help Manager also automatically adds a divider line between the end of the standard Help menu items and your appended items.

Listing 3-19 shows an 'hmnu' resource for the appended menu item shown in Figure 3-22.

Listing 3-19 Rez input for specifying help balloons for items in the Help menu

```
resource 'hmnu' (kHMHelpMenuID, "Help", purgeable) {
   HelpMgrVersion, 0, 0, 0,   /*header component*/
   HMSkipItem {               /*missing-items component*/
      /*no missing items, so skip to first appended menu-item */
      /* component*/
      },
   {     /*first menu-item component: SurfWriter Help command*/
      HMStringResItem { /*use an 'STR#' for help messages*/
      146, 1,  /*'STR#' res ID, index when item is enabled*/
      146, 2,  /*'STR#' res ID, index when item is dimmed*/
      146, 3,  /*'STR#' res ID, index when item is checked*/
      0,   0   /*item can't be marked*/
      },
   }
};
resource 'STR#' (146, "My help menu items' strings") {
   { /*array StringArray: six elements*/
   /*[1] enabled SurfWriter Help command help text*/
   "Offers tutorial help for the SurfWriter text processor.";
   /*[2] dimmed SurfWriter Help command help text*/
   "Offers tutorial help for the SurfWriter text processor. "
      "Not available until you open a SurfWriter document.";
   /*[3] checked SurfWriter Help command help text*/
   "Closes tutorial help for the SurfWriter text processor.";
   }
};
```

As previously explained in “Providing Help Balloons for Menus” beginning on page 3-27, the `'hmnu'` resource allows you to specify help balloons for four states of a menu item: enabled, dimmed, enabled and checked, and enabled and marked with a symbol other than a checkmark. You cannot specify a help balloon for a Help menu item that system software dims when an alert box or a modal dialog box appears, because you don’t have access to the missing-items component of the Help menu. When an alert box or a modal dialog box appears, the Help Manager displays a default help balloon for all dimmed Help menu items.

The Help Manager automatically processes the event when a user chooses any of the standard menu items in the Help menu. The Help Manager automatically enables and disables help when the user chooses Show Balloons or Hide Balloons from the Help menu. The setting of help is global and affects all applications.

The `MenuSelect` and `MenuKey` functions return a result with the menu ID in the high word and the menu item in the low word. Both functions return the `kHMHelpMenuID` constant (–16490) in the high word when the user chooses an appended item from the Help menu. The menu item number of the appended item is returned in the low word of the function result. The `DoMenuCommand` procedure shown in Listing 3-20 handles mouse clicks for those items defined by the application to appear in the Help menu.

Listing 3-20 Responding to the user’s choice in a menu command

```
PROCEDURE DoMenuCommand (menuResult: LongInt);
VAR
   menuID, menuItem: Integer;
BEGIN
   menuID := HiWrd(menuResult);     {get menu ID}
   menuItem := LoWrd(menuResult);   {get menu item number}
   CASE menuID OF
      mApple:        DoAppleMenuCommand(menuItem);
      mFile:         DoFileMenuCommand(menuItem);
      mEdit:         DoEditMenuCommand(menuItem);
      mFont:         DoFontMenuCommand(menuItem);
      kHMHelpMenuID: DoHelpMenuCommand(menuItem);
   END;
   HiliteMenu(0);
END;
```

In the future, Apple may choose to add other items to the Help menu. To determine the number of items in the Help menu, call the `CountMItems` function, which is described in the chapter "Menu Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

Writing Your Own Balloon Definition Function

The Help Manager takes care of positioning, sizing, and drawing your help balloons, and the standard balloon definition function provides a consistent and attractive shape to balloons across all applications.

Although it takes extra work on your part, and your balloons will not share the consistent appearance of help balloons used by the Finder and by other applications, you can create your own balloon definition function. The balloon definition function defines the appearance of the help balloon, which is a special type of window. You implement a balloon definition function by writing a window definition function that performs the tasks described in this section. The standard balloon definition function is of type `'WDEF'` with resource ID 126.

A balloon definition function is responsible for calculating the content region and structure region of the help balloon window and drawing the frame of the help balloon. The content region is the area inside the balloon frame; it contains the help message. The structure region is the boundary region of the entire balloon, including the content area and the pointer that extends from one of the help balloon's corners. (Figure 3-4 on page 3-10 illustrates the structure regions of the eight standard help balloons.)

The Help Manager first calculates the size of the rectangle that can enclose the help message and determines where to display the help balloon. The Help Manager uses TextEdit to determine any word and line breaks in the help message. The Help Manager determines where to display the help balloon based on the tip and alternate rectangle.

The Help Manager then adds a system-defined distance to the size of the rectangle. This distance allows for the tip of the help balloon. Note that the tip must always align with an edge of the boundary rectangle. The Help Manager uses the resulting rectangle as the boundary rectangle for the help balloon window.

To create the help balloon, the Help Manager uses the Window Manager function `NewWindow`. The Help Manager specifies the calculated rectangle and the window definition ID as parameters to `NewWindow`.

The `NewWindow` function calls the balloon definition function in the same manner as a window definition function. See the chapter "Window Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for more information on writing a window definition function.

The NewWindow function calls your balloon definition function with four parameters: the variation code that specifies the shape and relative tip position of the help balloon, a pointer to the window, the action to perform, and a parameter that has variable contents depending on the action to perform.

Here's an example that shows the declaration for a balloon definition function called MyBalloonDef.

```
FUNCTION MyBalloonDef (variant: Integer; theBalloon: WindowPtr;
                       message: Integer;
                       param: LongInt): LongInt;
```

The variant parameter is the variation code used to specify the shape and position of the help balloon. You should use the same relative position for the tip of the help balloon that the standard variation codes 0 through 7 specify (see Figure 3-4 on page 3-10). This ensures that the tip of the help balloon points to the object that the help balloon describes.

The parameter theBalloon is a pointer to the window of the help balloon.

The message parameter identifies the action your balloon definition function should perform. Your balloon definition function can be sent the same messages as a window definition function, but the only ones your balloon definition function needs to process are the wDraw and wCalcRgns messages.

When your balloon definition function receives the wCalcRgns message, your function should calculate the content region and structure region of the help balloon. When your balloon definition function receives the wDraw message, your function should draw the frame of the help balloon. If you want to process other messages in your balloon definition function (for example, performing any additional initialization), you can also process the other standard 'WDEF' messages. These messages, along with the wDraw and wCalcRgns messages, are described in the chapter "Window Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

The value of the param parameter depends on the value of the message parameter. The wCalcRgns and wDraw messages do not use this parameter.

If you want the Help Manager to use your balloon definition function, you specify its resource ID and the desired variation code either in the HMShowBalloon function or in the appropriate elements of the 'hmnu', 'hdlg', or 'hrct' resource. The Help Manager derives your balloon's window definition ID from its resource ID.

Help Manager Reference

This section describes the data structures, routines, and resources that are specific to the Help Manager.

The "Data Structures" section shows the data structures for the help message record and the Help Manager string list record. The "Help Manager Routines" section describes routines for determining Balloon Help status, displaying and removing help balloons, adding items to the Help menu, getting and setting the name and size of the font for help messages, setting and getting information for your application's help resources, determining the size of a help balloon, and getting the message of a help balloon. Should you want to replace the Help Manager's default balloon definition function and tip function with your own functions, the "Application-Defined Routines" section describes how. The "Resources" section describes the resources you can create to provide help balloons for your menus, alert and dialog boxes, and static windows; you can also create resources to override default help balloons provided by system software for various interface elements such as non-document Finder icons.

Data Structures

You can use two data structures to specify a help message to the `HMShowBalloon` function.

You use the help message record to describe the format and location of a help message. You specify the help message record as a parameter to the `HMShowBalloon` function.

If the message you want to pass to the `HMShowBalloon` function is stored in a string list (`'STR#'`) resource, you use a Help Manager string list record to specify the resource ID of a string list as well as an index to one of the strings in that list. You specify a Help Manager string list record in a field of a help message record.

The Help Message Record

A help message record describes a help message. The Help Manager displays a help balloon with that message when the help message record is passed in the `aHelpMsg` parameter to the `HMShowBalloon` function. The `HMMessageRecord` data type defines the help message record.

```
TYPE HMMessageRecord =
RECORD
   hmmHelpType:      Integer;             {type of next field}
   CASE Integer OF
      khmmString:    (hmmString: Str255); {Pascal string}
      khmmPict:      (hmmPict: Integer);  {'PICT' resource ID}
      khmmStringRes: (hmmStringRes: HMStringResType);
                                            {'STR#' resource ID }
                                            { and index}
      khmmTEHandle:  (hmmTEHandle: TEHandle);
                                            {TextEdit handle}
      khmmPictHandle:(hmmPictHandle: PicHandle);
                                            {picture handle}
      khmmTERes:     (hmmTERes: Integer);    {'TEXT'/'styl' }
                                            { resource ID}
      khmmSTRRes:    (hmmSTRRes: Integer)    {'STR ' resource ID}
END;
```

Field descriptions

`hmmHelpType` Specifies the data type of the next field of the help message record. You specify one of these constants for the `hmmHelpType` field.

```
CONST
khmmString     = 1; {Pascal string}
khmmPict       = 2; {'PICT' resource ID}
khmmStringRes  = 3; {'STR#' resource ID/index}
khmmTEHandle   = 4; {TextEdit handle}
khmmPictHandle = 5; {picture handle}
khmmTERes      = 6; {'TEXT'/'styl' resource ID}
khmmSTRRes     = 7; {'STR ' resource ID}
```

Only one field follows the `hmmHelpType` field, but it can be one of seven different data types. The field that follows the `hmmHelpType` field specifies the help message itself.

`hmmString` Contains a Pascal string for a help message when you supply the `khmmString` constant in the `hmmHelpType` field. (This is generally not recommended; instead, you should store the help message in a resource, which makes localization easier.)

`hmmPict` Contains the resource ID of a `'PICT'` resource for a help message when you supply the `khmmPict` constant in the `hmmHelpType` field.

`hmmStringRes` Contains a Help Manager string list record (described in “The Help Manager String List Record” on page 3-97) when you supply the `khmmStringRes` constant in the `hmmHelpType` field.

`hmmTEHandle` Specifies a TextEdit handle to a help message when you supply the `khmmTEHandle` constant in the `hmmHelpType` field.

`hmmPictHandle` Specifies a handle to a `'PICT'` graphic containing a help message when you supply the `khmmPictHandle` constant in the `hmmHelpType` field.

`hmmTERes`	Specifies the resource ID of both a `'TEXT'` and an `'styl'` resource for a help message when you supply the `khmmTEHandle` constant in the `hmmHelpType` field.
`hmmSTRRes`	Specifies the resource ID of an `'STR '` resource for a help message when you supply the `khmmSTRRes` constant in the `hmmHelpType` field.

Because the Help Manager uses the resource itself or the actual handle that you pass to `HMShowBalloon`, your `'PICT'` resource should be purgeable, or, when using a handle to a `'PICT'` resource, you should release the handle or dispose of it when you are finished with it.

Examples of how to use a help message record are provided in "Providing Help Balloons for Dynamic Windows" on page 3-74.

The Help Manager String List Record

To display a help message stored in an `'STR#'` resource with the `HMShowBalloon` function, use the `khmmStringRes` constant in the `hmmHelpType` field of the help message record (which you pass as a parameter to `HMShowBalloon`), and supply the `hmmStringRes` field of the help message record with a Help Manager string list record. (The help message record is described in the previous section.) The `HMStringResType` data type defines a Help Manager string list record.

```
TYPE HMStringResType =
RECORD
   hmmResID:   Integer; {'STR#' resource ID}
   hmmIndex:   Integer; {index of string}
END;
```

Field descriptions

`hmmResID`	Specifies the resource ID of the `'STR#'` resource.
`hmmIndex`	Specifies the index of a string within the `'STR#'` resource to use for a help message.

Help Manager Routines

This section describes the routines you use to display help balloons for the windows of your application. It also describes how to determine whether help is enabled; how to get the name and size of the text font in help balloons; how to set or override the help resources used with a menu, dialog box, or window; and how to get information about the window the help balloon is displayed in.

If you want to provide help balloons for the menus, alert boxes, dialog boxes, and static windows of your application, or if you want to override default help balloons provided by system software for various interface elements (such as non-document Finder icons), you only need to create the resources containing the descriptive information. "Using the Help Manager" beginning on page 3-18 gives details on how to create these resources.

If help is not enabled, most Help Manager routines do nothing and return the `hmHelpDisabled` result code.

IMPORTANT

All of the Help Manager routines may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt. Your application should not call Help Manager routines at interrupt time. ▲

Determining Balloon Help Status

The user turns on Balloon Help assistance by choosing Show Balloons from the Help menu. To determine whether help is currently enabled, you can use the `HMGetBalloons` function. If you display your own help balloons using the `HMShowBalloon` function, you should use the `HMGetBalloons` function to determine whether help is enabled before displaying a help balloon. If help is not enabled, you cannot display any help balloons. You can use the `HMIsBalloon` function to determine whether a help balloon is currently displayed on the screen.

HMGetBalloons

To determine whether Balloon Help assistance is enabled, use the `HMGetBalloons` function.

```
FUNCTION HMGetBalloons: Boolean;
```

DESCRIPTION

The `HMGetBalloons` function returns `TRUE` if help is currently enabled and `FALSE` if help is not currently enabled. Because the `HMGetBalloons` function does not load the Help Manager into memory, it provides a fast way to determine whether Balloon Help assistance is enabled.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMGetBalloons` function are

Trap macro	Selector
`_Pack14`	$0003

SEE ALSO

To determine whether Balloon Help assistance is available, use the `Gestalt` function as described in “Using the Help Manager” on page 3-18.

HMIsBalloon

To determine whether the Help Manager is currently displaying a help balloon, use the `HMIsBalloon` function.

```
FUNCTION HMIsBalloon: Boolean;
```

DESCRIPTION

The `HMIsBalloon` function returns `TRUE` if a help balloon is currently displayed on the screen and `FALSE` if a help balloon is not currently displayed. This function is useful for determining whether a balloon is showing before you redraw the screen. For example, you might want to determine whether a balloon is displayed so that you can remove it before opening or closing a window.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMIsBalloon` function are

Trap macro	Selector
`_Pack14`	$0007

Displaying and Removing Help Balloons

When the user turns on Balloon Help assistance, the Help Manager automatically tracks the cursor and displays and removes help balloons as the cursor moves over hot rectangles specified in `'hrct'` resources or over display rectangles associated with menu items specified in `'hmnu'` resources and items specified in `'hdlg'` resources. If you want to provide help balloons for areas not defined in these resources, then your application is responsible for tracking the cursor and displaying and removing balloons for these application-defined areas.

To display a help balloon in your application-defined area, use the `HMShowBalloon` function. If your application uses its own menu definition procedure, use the `HMShowMenuBalloon` function to display a balloon described by the standard balloon definition function. To remove a balloon that you display using `HMShowMenuBalloon`, you must use the `HMRemoveBalloon` function. To remove a balloon that you display using `HMShowBalloon`, you can either use the `HMRemoveBalloon` function to remove the help balloon, or you can let the Help Manager remove it for you.

HMShowBalloon

To display a help balloon of the content area of any window of your application, you can use the `HMShowBalloon` function. If the user has enabled Balloon Help assistance, the `HMShowBalloon` function displays a help balloon containing the message specified by the `aHelpMsg` parameter.

```
FUNCTION HMShowBalloon (aHelpMsg: HMMessageRecord;
                        tip: Point; alternateRect: RectPtr;
                        tipProc: Ptr; theProc, variant: Integer;
                        method: Integer): OSErr;
```

`aHelpMsg`	The message displayed in the help balloon.
`tip`	The location, in global coordinates, of the help balloon's tip.
`alternateRect`	A rectangle, in global coordinates, that the Help Manager uses if necessary to calculate a new tip location. If you specify a rectangle in this parameter, the Help Manager automatically calls the `HMRemoveBalloon` function to remove the help balloon when the user moves the cursor outside the area bounded by the rectangle. If you instead pass `NIL` in this parameter, your application must use the `HMRemoveBalloon` function to remove the help balloon when appropriate.
`tipProc`	The tip function called by the Help Manager before displaying the balloon. Specify `NIL` to use the Help Manager's default tip function, or supply your own tip function and point to it in this parameter.
`theProc`	The balloon definition function. To use the standard balloon definition function, specify 0 in this parameter. To use your own balloon definition function, specify the resource ID of the `'WDEF'` resource containing your balloon definition function.
`variant`	The variation code for the balloon definition function. Specify 0 in the `variant` parameter to use the default help balloon position, specify a code from 1 to 7 to use one of the other positions provided by the standard balloon definition function, or specify another code to use one of the positions provided by your own balloon definition function.
`method`	A value that indicates whether the Help Manager should save the bits behind the balloon and whether to generate an update event. You can pass one of the following constants in this parameter: `kHMRegularWindow`, `kHMSaveBitsNoWindow`, or `kHMSaveBitsWindow`.

DESCRIPTION

If help is enabled, the `HMShowBalloon` function displays a help balloon with the help message you specify in the `aHelpMsg` parameter. You use global coordinates to specify the tip and the rectangle pointed to by the `alternateRect` parameter. The Help Manager calculates the location and size of the help balloon. If it fits onscreen, the Help Manager displays the help balloon using the specified tip location.

If you use the `HMShowBalloon` function to display help balloons, you must identify hot rectangles, create your own data structures to store their locations, track the cursor yourself, and call `HMShowBalloon` when the cursor moves to your hot rectangles. The Help Manager does not know the locations of your hot rectangles, so it cannot use them for moving the tip if the help balloon is placed offscreen. Instead, the Help Manager uses the alternate rectangle that you point to with the `alternateRect` parameter. Often, you specify the same coordinates for the alternate rectangle that you specify for your hot rectangle. However, you may choose to make your alternate rectangle smaller or larger than your hot rectangle. If you make your alternate rectangle smaller than your hot rectangle, you have greater assurance that the Help Manager will be able to fit the help balloon onscreen; if you specify an alternate rectangle that is larger than your hot rectangle, you have greater assurance that the balloon will not obscure the object it explains.

If you specify a rectangle in the `alternateRect` parameter, the Help Manager automatically calls `HMRemoveBalloon` to remove the balloon when the cursor leaves the area bounded by the rectangle.

If the balloon's first position is partly offscreen or if it intersects the menu bar, the Help Manager tries a combination of different balloon variation codes and different tip positions along the sides of the alternate rectangle to make the balloon fit. Figure 3-5 on page 3-11 shows what happens when the balloon's first two positions are located offscreen. If, after exhausting all possible positions, the Help Manager cannot fit the entire balloon onscreen, the Help Manager displays a balloon at the position that best fits onscreen and clips the help message to fit at this position. If the coordinates specified by both the original tip and the `alternateRect` parameter are offscreen, the Help Manager does not display the balloon at all.

If you specify `NIL` for the `alternateRect` parameter, your application is responsible for tracking the cursor and determining when to remove the balloon. The Help Manager also does not attempt to calculate a new tip location if the balloon is offscreen.

Once the Help Manager determines the location and size of the help balloon, the Help Manager calls the function pointed to by the `tipProc` parameter before displaying the balloon. Specify `NIL` in the `tipProc` parameter to use the Help Manager's default tip function.

You can supply your own tip function and point to it in the `tipProc` parameter. The Help Manager calls the tip function after calculating the location of the balloon and before displaying it. In the parameters of your tip function, the Help Manager returns the tip, the region boundary of the entire balloon, the region boundary for the content area within the balloon frame, and the variation code to be used for the balloon. This allows you to examine and possibly adjust the balloon before it is displayed.

The Help Manager reads the balloon definition function specified by the parameter `theProc` into memory if it isn't already in memory. If the balloon definition function can't be read into memory, the help balloon is not displayed and the `HMShowBalloon` function returns the `resNotFound` result code.

The `method` parameter specifies whether the Help Manager should save the bits behind the balloon and whether to generate an update event. You can supply one of these constants for the parameter.

```
CONST kHMRegularWindow     = 0;  {don't save bits; just update}
      kHMSaveBitsNoWindow  = 1;  {save bits; don't do update}
      kHMSaveBitsWindow    = 2;  {save bits; do update event}
```

If you specify `kHMRegularWindow`, the Help Manager draws and removes the help balloon as if it were a window. That is, when drawing the balloon, the Help Manager does not save bits behind the balloon, and, when removing the balloon, the Help Manager generates an update event. This is the standard behavior of help balloons; it is the behavior you should normally use.

If you specify `kHMSaveBitsNoWindow` in the `method` parameter, the Help Manager does not create a window for displaying the balloon. Instead, the Help Manager creates a help balloon that is more like a menu than a window. The Help Manager saves the bits behind the balloon when it creates the balloon. When it removes the balloon, the Help Manager restores the bits without generating an update event. You should use this method only in a modal environment where the bits behind the balloon cannot change from the time the balloon is drawn to the time it is removed. For example, you might specify the `kHMSaveBitsNoWindow` constant when providing help balloons for pop-up menus that overlay complex graphics, which might take a long time to redraw with an update event.

If you specify `kHMSaveBitsWindow`, the Help Manager treats the help balloon as a hybrid having properties of both a menu and a window. That is, the Help Manager saves the bits behind the balloon when it creates the balloon, and, when it removes the balloon, it both restores the bits and generates an update event. You'll rarely need this option. It is necessary only in a modal environment that might immediately change to a nonmodal environment—that is, where the bits behind the balloon are static when the balloon is drawn, but can possibly change before the balloon is removed.

`HMShowBalloon` returns the `noErr` result code if the help balloon was successfully displayed.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and the routine selector for the `HMShowBalloon` function are

Trap macro	Selector
`_Pack14`	$0B01

RESULT CODES

`noErr`	0	No error; the help balloon was displayed
`paramErr`	–50	Error in parameter list
`memFullErr`	–108	Not enough room in heap zone
`resNotFound`	–192	Unable to read resource
`hmHelpDisabled`	–850	Help balloons are not enabled
`hmBalloonAborted`	–853	Because of constant cursor movement, the help balloon wasn't displayed
`hmOperationUnsupported`	–861	Invalid value passed in the `method` parameter

SEE ALSO

You specify the help message in the `aHelpMsg` parameter. "Providing Help Balloons for Dynamic Windows" beginning on page 3-74 shows how to specify this information.

You can supply your own tip function (as explained in the description of the `MyTip` function, which begins on page 3-130) and point to it in the `tipProc` parameter.

Figure 3-4 on page 3-10 illustrates the variation codes you can specify in the `variant` parameter and their corresponding help balloon positions for the standard balloon definition function.

If your application uses its own menu definition procedure, you can use the `HMShowMenuBalloon` function to display help balloons for the menus that your menu definition procedure manages. The `HMShowMenuBalloon` function is next.

HMShowMenuBalloon

The Help Manager displays help balloons for applications that provide `'hmnu'` resources and use the standard menu definition procedure. If your application uses your own menu definition procedure, you can still use the Help Manager to display help balloons for the menus that your menu definition procedure manages. Use the `HMShowMenuBalloon` function to display balloons described by the standard balloon definition function. If you want to use your own balloon definition function from within your menu definition procedure, call the `HMShowBalloon` function (described in the previous section) and specify the `kHMSaveBitsNoWindow` constant for the `method` parameter. You can also use the `HMShowMenuBalloon` function as an alternative to creating an `'hmnu'` resource for your menu.

```
FUNCTION HMShowMenuBalloon (itemNum: Integer; itemMenuID: Integer;
                            itemFlags: LongInt;
                            itemReserved: LongInt;
                            tip: Point; alternateRect: RectPtr;
                            tipProc: Ptr; theProc: Integer;
                            variant: Integer): OSErr;
```

`itemNum`	The number of the menu item over which the cursor is currently located. Use a positive number in the `itemNum` parameter to specify a menu item, use –1 if the cursor is located over a divider line, or use 0 if the cursor is located over the menu title.
`itemMenuID`	The ID of the menu in which the cursor is currently located.
`itemFlags`	A long integer from the menu flags, telling whether a menu item is enabled or dimmed and whether the menu itself is enabled or dimmed. The Help Manager uses this value to determine which balloon to display from the `'hmnu'` resource.
`itemReserved`	Reserved for future use by Apple. Specify 0 in this parameter.
`tip`	The tip for the help balloon. The standard menu definition procedure places the tip 8 pixels from either the right or left edge of the menu item. For menu titles, the standard menu definition procedure centers the tip at the bottom of the menu bar; you should not specify a tip with coordinates in the menu bar for any menu titles. The Help Manager uses the tip you specify in this parameter unless it places the help balloon offscreen or in the menu bar. If the tip is offscreen, the Help Manager uses the rectangle specified in the `alternateRect` parameter to calculate a new tip location.
`alternateRect`	The rectangle that the Help Manager uses to calculate a new tip location. (The standard menu definition procedure specifies the alternate rectangle as the rectangle that encloses the menu title or menu item.) If the balloon's first position is offscreen or in the menu bar, the Help Manager tries a different balloon variation code or calculates a new tip by transposing it to an opposite side of the alternate rectangle. If you specify `NIL` for the `alternateRect` parameter, the Help Manager does not attempt to calculate a new tip position when the help balloon is offscreen.
`tipProc`	The tip function that the Help Manager calls before displaying the balloon. Specify `NIL` to use the Help Manager's default tip function, or supply your own tip function and point to it in this parameter.
`theProc`	Reserved for use by Apple. Specify 0 in this parameter.
`variant`	The variation code for the standard balloon definition function. Specify 0 to use the default balloon position or a code between 1 and 7 to use one of the other standard positions shown in Figure 3-4 on page 3-10.

DESCRIPTION

The `HMShowMenuBalloon` function saves the bits behind the help balloon before displaying the help balloon. When you remove the balloon, the Help Manager restores the bits that were previously behind it.

After your menu definition procedure determines that the cursor is located in a menu item, you can use the HMShowMenuBalloon function to display any help balloons associated with that item. You must then use the HMRemoveBalloon function to remove the balloon when the cursor moves away from the menu item.

If you use the HMShowMenuBalloon function to display help balloons, you must identify hot rectangles, create your own data structures to store their locations, track the cursor yourself, and call HMShowMenuBalloon when the cursor moves to your hot rectangles. The Help Manager does not know the locations of your hot rectangles, so it cannot use them for moving the tip if the balloon is placed offscreen. Instead, the Help Manager uses the alternate rectangle that you point to with the alternateRect parameter.

Unlike the way the alternateRect parameter works in the HMShowBalloon function, specifying an alternate rectangle to HMShowMenuBalloon does not cause the Help Manager to track the cursor and remove the balloon for you. You must still track the cursor and use the HMRemoveBalloon function to remove the balloon when the cursor moves out of the area specified by the hot rectangle.

Specify NIL in the tipProc parameter to use the tip function values calculated by the Help Manager. If you supply your own tip function and specify it in the tipProc parameter, the Help Manager returns the tip, the region boundary of the entire balloon, the region boundary for the content area within the balloon frame, and the variation code to be used for the help balloon before displaying it. This allows you to examine and possibly adjust the balloon before it is displayed.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the HMShowMenuBalloon function are

Trap macro	**Selector**
_Pack14	$0E05

RESULT CODES

noErr	0	No error; the help balloon was displayed
memFullErr	–108	Not enough room in heap zone
hmHelpDisabled	–850	Help balloons are not enabled
hmBalloonAborted	–853	Because of constant cursor movement, the help balloon wasn't displayed
hmSameAsLastBalloon	–854	Menu and item are same as last menu and item

SEE ALSO

You can supply your own tip function (as explained in the description of the MyTip function, which begins on page 3-130) and point to it in the tipProc parameter.

The HMRemoveBalloon function is described next.

HMRemoveBalloon

To remove a help balloon that your application displays using the function HMShowMenuBalloon, use the HMRemoveBalloon function. If your application does not specify an alternate rectangle to the HMShowBalloon function, use HMRemoveBalloon to remove the help balloon you display with HMShowBalloon.

```
FUNCTION HMRemoveBalloon: OSErr;
```

DESCRIPTION

The HMRemoveBalloon function removes any balloon that is currently visible—unless the user is using Close View and is pressing the Shift key. (This action keeps the help balloon onscreen even while the user moves away from the hot rectangle under Close View.)

If you use the HMShowBalloon function to display help balloons, you can either let the Help Manager track the cursor and remove the balloon when the cursor moves out of the hot rectangle, or your application can track the cursor and determine when to remove the balloon. To let the Help Manager track the cursor and remove the balloon when using the HMShowBalloon function, specify a rectangle in the alternateRect parameter. If you want your application to track the cursor and remove the balloon when using the HMShowBalloon function, specify NIL in the alternateRect parameter. You must then use the HMRemoveBalloon function to remove the balloon when the user moves the cursor outside the rectangle.

If you use the HMShowMenuBalloon function to display help balloons, you must always track the cursor and use the HMRemoveBalloon function to remove the balloon when the cursor moves out of the hot rectangle.

▲ **WARNING**
The HMRemoveBalloon function removes any help balloon that is currently visible, regardless of the application that displayed it. You should call HMRemoveBalloon only when the cursor is in the content area of your application window but not in a hot rectangle, and you should never call it when your application is in the background. ▲

If the user is using Close View and is pressing the Shift key, the help balloon stays onscreen even while the user moves away from the hot rectangle. The HMRemoveBalloon function returns a result code of hmCloseViewActive in this case.

If you use your own menu definition procedure, you should call HMRemoveBalloon when your procedure receives messages about saving or restoring bits. (These messages are described in the chapter “Menu Manager” in *Inside Macintosh: Macintosh Toolbox Essentials*.)

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMRemoveBalloon` function are

Trap macro	Selector
`_Pack14`	$0002

RESULT CODES

`noErr`	0	No error or the help balloon was removed
`hmHelpDisabled`	–850	Help balloons are not enabled
`hmNoBalloonUp`	–862	No balloon showing
`hmCloseViewActive`	–863	Balloon can't be removed because Close View is in use

SEE ALSO

The description of the `HMShowBalloon` function begins on page 3-100; the description of the `HMShowMenuBalloon` function begins on page 3-103.

Enabling and Disabling Balloon Help Assistance

You can enable or disable help using the `HMSetBalloons` function. If you enable or disable help, you do so for all applications. Because the setting of Balloon Help assistance should be under the user's control, in most cases you should not modify the user's setting. However, if you feel your application absolutely must enable or disable Balloon Help assistance, you can use the `HMSetBalloons` function. If you modify this setting, return it to its previous state as soon as possible.

HMSetBalloons

To enable or disable Balloon Help assistance for the user, use the `HMSetBalloons` function.

```
FUNCTION HMSetBalloons (flag: Boolean): OSErr;
```

`flag`	Specifies whether help should be enabled or disabled for all applications and the system software.

DESCRIPTION

If the value of the `flag` parameter is `TRUE`, `HMSetBalloons` enables Balloon Help assistance. If the value of the `flag` parameter is `FALSE`, `HMSetBalloons` disables Balloon Help assistance. If a help balloon is showing, you must first remove it using the `HMRemoveBalloon` function before you use `HMSetBalloons` to disable Balloon Help assistance.

SPECIAL CONSIDERATIONS

When Balloon Help assistance is disabled, the Help Manager does not display help balloons for any applications. When help is disabled, the `HMShowBalloon` and `HMShowMenuBalloon` functions do not display help balloons; they return nonzero result codes.

Because the setting of Balloon Help assistance should be under the user's control, you generally should not use the `HMSetBalloons` function.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMSetBalloons` function are

Trap macro	Selector
`_Pack14`	$0104

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in parameter list
`memFullErr`	–108	Not enough room in heap zone
`resNotFound`	–192	Unable to read resource

SEE ALSO

The description of the `HMShowBalloon` function begins on page 3-100; the description of the `HMShowMenuBalloon` function begins on page 3-103.

Adding Items to the Help Menu

The Help Manager automatically appends the Help menu when your application inserts an Apple menu into its menu bar. The Menu Manager automatically appends the Help menu to the right of all your menus and to the left of the Application menu (and to the left of the Keyboard menu if a non-Roman script system is installed).

The Help menu is specific to each application. The Help menu items defined by the Help Manager should be common to all applications, but you can append your own menu items for help-related information by using the `HMGetHelpMenuHandle` function.

HMGetHelpMenuHandle

To append items to the Help menu, use the `HMGetHelpMenuHandle` function.

```
FUNCTION HMGetHelpMenuHandle (VAR mh: MenuHandle): OSErr;
```

`mh`	A copy of a handle to the Help menu.

DESCRIPTION

The HMGetHelpMenuHandle function returns in its mh parameter a handle to your application's help menu. With this handle, you can append items to the Help menu by using the AppendMenu procedure or other related Menu Manager routines. The Help Manager automatically adds the divider line that separates your items from the rest of the Help menu.

Be sure to define help balloons for your items in the Help menu by creating an 'hmnu' resource and specifying the kHMHelpMenuID constant as its resource ID.

The Menu Manager functions MenuSelect and MenuKey return a result with the menu ID in the high word and the menu item in the low word. Both functions return the HelpMgrID constant in the high word when the user chooses an appended item from the Help menu. The number of the appended menu item is returned in the low word of the function result. In the future, Apple Computer, Inc., may choose to add other items to the Help menu. To determine the number of items in the Help menu, call the Menu Manager function CountMItems.

SPECIAL CONSIDERATIONS

Do not use the Menu Manager function GetMenuHandle to get a handle to the Help menu, because GetMenuHandle returns a handle to the global Help menu, not the Help menu that is specific to your application.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the HMGetHelpMenuHandle function are

Trap macro	Selector
_Pack14	$0200

RESULT CODES

noErr	0	No error
paramErr	–50	Error in parameter list
memFullErr	–108	Not enough room in heap zone
resNotFound	–192	Unable to read resource
hmHelpManagerNotInited	–855	Help menu not set up

SEE ALSO

"Adding Menu Items to the Help Menu" beginning on page 3-90 provides details and illustrative sample code for using HMGetHelpMenuHandle. The 'hmnu' resource is described in detail in "Providing Help Balloons for Menus" beginning on page 3-27. See the chapter "Menu Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for information about AppendMenu, MenuSelect, MenuKey, and other Menu Manager routines.

Getting and Setting the Font Name and Size

Using the `HMGetFont` and `HMGetFontSize` functions, you can get information about the font name and size currently used for text strings displayed in help balloons. Using the `HMSetFont` and `HMSetFontSize` functions, you can change the font name and size.

HMGetFont

To get information about the font that is currently used to display text in help balloons, use the `HMGetFont` function.

```
FUNCTION HMGetFont (VAR font: Integer): OSErr;
```

`font`	The global font number used to display text in help balloons.

DESCRIPTION

The `HMGetFont` function returns in its `font` parameter the global font number used to display text in help balloons. `HMGetFont` returns this information only for Pascal strings stored in the help resources themselves and for strings from `'STR#'` and `'STR '` resources; it does not return information about text in `'PICT'` or styled text resources, or in handles to either of these resources.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMGetFont` function are

Trap macro	Selector
`_Pack14`	$020A

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Not enough room in heap zone

SEE ALSO

The chapter “TextEdit” in *Inside Macintosh: Text* describes global font numbers.

HMGetFontSize

To get information about the font size that is currently used to display text in help balloons, use the `HMGetFontSize` function.

```
FUNCTION HMGetFontSize (VAR fontSize: Integer): OSErr;
```

`fontSize`	The global font size used to display text in help balloons.

DESCRIPTION

The `HMGetFontSize` function returns in its `fontSize` parameter the global font size used to display text in help balloons. This information applies only to Pascal strings stored in the help resources themselves and to strings from `'STR#'` and `'STR '` resources; it does not apply to text in `'PICT'` or styled text resources, or in handles to either of these resources.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMGetFontSize` function are

Trap macro	Selector
`_Pack14`	$020B

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Not enough room in heap zone

SEE ALSO

See the chapter “TextEdit” in *Inside Macintosh: Text* for detailed information about font sizes.

HMSetFont

You can use the `HMSetFont` function to specify the font used to display text in help balloons.

```
FUNCTION HMSetFont (font: Integer): OSErr;
```

`font`	A global font number.

DESCRIPTION

The `HMSetFont` function sets the font for help balloons in all applications that display help balloons.

This function applies only to Pascal strings stored in the help resources themselves and to strings from `'STR#'` and `'STR '` resources; it does not apply to text in `'PICT'` or styled text resources, or in handles to either of these resources.

SPECIAL CONSIDERATIONS

Use this function with extreme restraint, because the default font provides a consistent look across applications. If your application uses this function to change the font name or size, the change affects all applications that display help balloons.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMSetFont` function are

Trap macro	Selector
`_Pack14`	$0108

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Not enough room in heap zone

SEE ALSO

See the chapter “TextEdit” in *Inside Macintosh: Text* for detailed information about fonts and font numbers.

HMSetFontSize

You can use the `HMSetFontSize` function to specify the font size used to display text in help balloons.

```
FUNCTION HMSetFontSize (fontSize: Integer): OSErr;
```

`fontSize`	The global font size the Help Manager uses to display text in help balloons.

DESCRIPTION

The `HMSetFontSize` function sets the font size for help balloons in all applications and software that display help balloons. This function applies only to Pascal strings stored in the help resources themselves and to strings from `'STR#'` and `'STR '` resources; it does not apply to text in `'PICT'` or styled text resources, or in handles to either of these resources.

SPECIAL CONSIDERATIONS

Use this function with extreme restraint, because the default font size provides a consistent look across applications. If your application uses this function to change the font size, the change affects all applications that display help balloons.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMSetFontSize` function are

Trap macro	Selector
`_Pack14`	$0109

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Not enough room in heap zone

SEE ALSO

See the chapter “TextEdit” in *Inside Macintosh: Text* for detailed information about fonts and font sizes.

Setting and Getting Information for Help Resources

Using the `HMSetMenuResID` or `HMScanTemplateItems` function, you can set help resources for menus, dialog boxes, or windows of your application that do not currently have help resources associated with them. You can also supplement the `'hmnu'` and `'hdlg'` resources currently associated with the menus and dialog boxes of your application by using the `HMSetMenuResID` or `HMSetDialogResID` function. You can use the `HMGetMenuResID` function to determine the `'hmnu'` resource ID associated with a menu.

When you use the `HMSetDialogResID` function, you can supplement any `'hdlg'` resources that are specified in item list (`'DITL'`) resources. The resource you specify in the `HMSetDialogResID` function adds to any help that already exists in the form of an `'hdlg'` resource for the next dialog box or alert box to be displayed. You can use an `'hdlg'` resource (described in "Providing Help Balloons for Items in Dialog Boxes and Alert Boxes" on page 3-51) to provide help balloons for items common to several dialog boxes and alert boxes, and you can use the `HMSetDialogResID` function to provide help balloons for items that you add to individual dialog boxes and alert boxes.

You can use the `HMGetDialogResID` function to get the resource ID of the `'hdlg'` resource that will be used by the next dialog box as a result of a previous call to the `HMSetDialogResID` function. If the `'hdlg'` resource currently in use has not been overridden by a call to `HMSetDialogResID`, the `HMGetDialogResID` function returns a result code of `resNotFound`.

You can use the `HMGetDialogResID` and `HMSetDialogResID` functions when displaying nested dialog boxes (although, in general, you should close one dialog box before displaying another). For example, you can save the `'hdlg'` resource of the current dialog box, set a new `'hdlg'` resource, display the new dialog box, and then restore the setting of the previous `'hdlg'` resource when you close the second dialog box.

HMSetMenuResID

You can use the `HMSetMenuResID` function to set the `'hmnu'` resource for a menu that did not previously have one or to supplement the existing `'hmnu'` resource for a menu.

```
FUNCTION HMSetMenuResID (menuID, resID: Integer): OSErr;
```

`menuID`	The menu to associate with the `'hmnu'` resource.
`resID`	The resource ID of the `'hmnu'` resource to use for the menu specified by the `menuID` parameter.

DESCRIPTION

The `resID` parameter specifies the resource ID of the `'hmnu'` resource to use for the menu specified by the `menuID` parameter. The menu identified by the `menuID` parameter should correspond to an existing menu in your menu list. The Help Manager maintains a list of the menus whose `'hmnu'` resources you set using the `HMSetMenuResID` function.

Before your application terminates, specify –1 in the `resID` parameter to disassociate a particular menu and an `'hmnu'` resource that you previously associated using the `HMSetMenuResID` function.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMSetMenuResID` function are

Trap macro	Selector
`_Pack14`	$020D

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Not enough room in heap zone

SEE ALSO

"Providing Help Balloons for Menus You Disable for Dialog Boxes" beginning on page 3-47 describes how to use `HMSetMenuResID` to associate an alternate `'hmnu'` resource with a menu that your application dims when it displays a dialog box.

HMGetMenuResID

After you use the `HMSetMenuResID` function to associate a menu with an `'hmnu'` resource, you can use the `HMGetMenuResID` function to get the resource ID of the `'hmnu'` resource.

```
FUNCTION HMGetMenuResID (menuID: Integer;
                         VAR resID: Integer): OSErr;
```

`menuID`	The menu for which you want the associated `'hmnu'` resource. The value specified in the `menuID` parameter must have been previously associated using the `HMSetMenuResID` function.
`resID`	The resource ID of the `'hmnu'` resource associated with the specified menu.

DESCRIPTION

HMGetMenuResID returns in its resID parameter the resource ID of the 'hmnu' resource associated with the menu specified by the menuID parameter. If the menu does not have an 'hmnu' resource that was previously set using HMSetMenuResID, the HMGetMenuResID function returns –1 in the resID parameter and a nonzero result code.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the HMGetMenuResID function are

Trap macro	Selector
_Pack14	$0314

RESULT CODES

noErr	0	No error
resNotFound	–192	Unable to read resource

SEE ALSO

The HMSetMenuResID function is described on page 3-114.

HMScanTemplateItems

You can use the HMScanTemplateItems function to search for a resource of type 'hdlg' or 'hrct'.

```
FUNCTION HMScanTemplateItems (whichID, whichResFile: Integer;
                              whichType: ResType): OSErr;
```

whichID	The resource ID of the 'hdlg' or 'hrct' resource to search for.
whichResFile	The file reference number of the resource file to search.
whichType	The type of help resource to search for—either 'hdlg' or 'hrct'.

DESCRIPTION

The HMScanTemplateItems function searches a resource file for resources of type 'hdlg' or 'hrct'. Specify the resource ID of the 'hdlg' or 'hrct' resource to search for in the whichID parameter. Specify the resource type in the whichType parameter. When HMScanTemplateItems returns the value for noErr, the Help Manager applies the help messages in the specified 'hdlg' or 'hrct' resource to the active window.

The resource file specified in the `whichResFile` parameter must already be open. Specify –1 in the `whichResFile` parameter to search the current resource file.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMScanTemplateItems` function are

Trap macro	**Selector**
`_Pack14`	$0410

RESULT CODES

`noErr`	0	No error
`fnOpnErr`	–38	File not open
`memFullErr`	–108	Not enough room in heap zone
`resNotFound`	–192	Unable to read resource

SEE ALSO

If you want the capability that `HMScanTemplateItems` provides without modifying your code, you can add a `HelpItem` item to your item list (`'DITL'`) resources or add an `'hwin'` resource—as described in “Using a Help Item Versus Using an 'hwin' Resource” on page 3-63 and in “Associating Help Resources With Static Windows” on page 3-68.

HMSetDialogResID

You can use the `HMSetDialogResID` function to set the `'hdlg'` resource that specifies help balloons for the next dialog box or alert box.

```
FUNCTION HMSetDialogResID (resID: Integer): OSErr;
```

`resID` The resource ID of the `'hdlg'` resource to use when your application displays the next dialog box or alert box.

DESCRIPTION

The `HMSetDialogResID` function uses the `'hdlg'` resource specified in the `resID` parameter to supplement whatever `'hdlg'` resource might already be associated with the next dialog box or alert box that you display. `HMSetDialogResID` supplements the help messages specified by a `HelpItem` item in the next dialog or alert box’s item list (`'DITL'`) resource. Specify –1 in the `resID` parameter to reset or clear a previous call to the `HMSetDialogResID` function.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMSetDialogResID` function are

Trap macro	**Selector**
`_Pack14`	$010C

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Not enough room in heap zone

SEE ALSO

You typically use `HMSetDialogResID` in conjunction with the `HMGetDialogResID` function, which is described in the following section.

HMGetDialogResID

You can use the `HMGetDialogResID` function to get the resource ID of the `'hdlg'` resource that will be used by the next dialog box as a result of a previous call to the `HMSetDialogResID` function.

```
FUNCTION HMGetDialogResID (VAR resID: Integer): OSErr;
```

`resID`	The resource ID of the last `'hdlg'` resource set with the `HMSetDialogResID` function.

DESCRIPTION

The `HMGetDialogResID` function returns in its `resID` parameter the resource ID of the last `'hdlg'` resource set with the `HMSetDialogResID` function.

You can use the `HMGetDialogResID` and `HMSetDialogResID` functions when your application displays nested dialog boxes (although you should generally close one dialog box before displaying another). For example, you can save the `'hdlg'` resource of the current dialog box, set a new `'hdlg'` resource, display the new dialog box, and then restore the setting of the previous `'hdlg'` resource when you close the second dialog box.

If the `'hdlg'` resource currently in use was not set by a call to the `HMSetDialogResID` function, the `HMGetDialogResID` function returns a result code of `resNotFound`.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMGetDialogResID` function are

Trap macro	**Selector**
`_Pack14`	$0213

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Not enough room in heap zone
`resNotFound`	–192	Unable to read resource

SEE ALSO

You typically use `HMGetDialogResID` in conjunction with the `HMSetDialogResID` function, which is described on page 3-117.

Determining the Size of a Help Balloon

If your application does extensive drawing, the Help Manager provides three functions that may be helpful for determining the dimensions of your help balloons before displaying them. Then you can ensure that your help balloons don't obscure an area that requires an inordinate amount of time to update.

To get the size of a help balloon before the Help Manager displays it, use the `HMBalloonRect` or `HMBalloonPict` function. To get the size of the currently displayed help balloon, use the `HMGetBalloonWindow` function.

HMBalloonRect

To get information about the size of a help balloon before the Help Manager displays it, you can use the `HMBalloonRect` function.

```
FUNCTION HMBalloonRect (aHelpMsg: HMMessageRecord;
                        VAR coolRect: Rect): OSErr;
```

`aHelpMsg`	The help message for the help balloon.
`coolRect`	The coordinates of the rectangle that encloses the help message. The upper-left corner of the rectangle has the coordinates (0,0).

DESCRIPTION

The `HMBalloonRect` function calculates the coordinates that the Help Manager uses for a particular balloon, permitting you to specify the help message for a help balloon and then obtaining the size (but not the position) of the rectangle used for the balloon. Note that the `HMBalloonRect` function does not display the help balloon.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMBalloonRect` function are

Trap macro	Selector
`_Pack14`	$040E

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in parameter list
`memFullErr`	–108	Not enough room in heap zone

SEE ALSO

The `aHelpMsg` parameter is of data type `HMMessageRecord`, which is described in “Providing Help Balloons for Dynamic Windows” beginning on page 3-74.

HMBalloonPict

To get a handle to a picture before displaying it in a help balloon, use the `HMBalloonPict` function.

```
FUNCTION HMBalloonPict (aHelpMsg: HMMessageRecord;
                        VAR coolPict: PicHandle): OSErr;
```

`aHelpMsg`	The help message for the help balloon; in this case, a picture.
`coolPict`	A handle to the picture that the Help Manager will use if you later choose to display the help balloon.

DESCRIPTION

The `HMBalloonPict` function does not display the help balloon; it returns a handle to the picture that the Help Manager will use if you later choose to display a help balloon with the specified help message.

The `pictFrame` field of the picture handle in the `coolPict` parameter contains the same rectangle as the rectangle obtained from the `HMBalloonRect` function. The rectangle specifies the display rectangle that surrounds the picture.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMBalloonPict` function are

Trap macro	**Selector**
`_Pack14`	$040F

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in parameter list
`memFullErr`	–108	Not enough room in heap zone

SEE ALSO

The `aHelpMsg` parameter is of data type `HMMessageRecord`. "Providing Help Balloons for Dynamic Windows" beginning on page 3-74 describes the fields of this record.

HMGetBalloonWindow

The Help Manager displays help balloons in special windows; to get a pointer to the window record of the currently displayed help balloon, use the `HMGetBalloonWindow` function.

```
FUNCTION HMGetBalloonWindow (VAR window: WindowPtr): OSErr;
```

`window`	A pointer to the window record for the currently displayed help balloon.

DESCRIPTION

In its `window` parameter, `HMGetBalloonWindow` returns a pointer to the window record for the currently displayed help balloon. The window record contains a graphics port record, which in turn defines the port's rectangle.

If no help balloon is currently displayed, the `HMGetBalloonWindow` function returns `NIL` in the `window` parameter. The `HMGetBalloonWindow` function also returns `NIL` for balloons created with the `HMShowMenuBalloon` function because no windows are created; likewise, `NIL` is returned for balloons created with the `HMShowBalloon` function when the `kHMSaveBitsNoWindow` constant is specified as the `method` parameter.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMGetBalloonWindow` function are

Trap macro	**Selector**
`_Pack14`	$0215

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Not enough room in heap zone

SEE ALSO

The description of the `HMShowMenuBalloon` function begins on page 3-103; the description of the `HMShowBalloon` function begins on page 3-100.

Getting the Message of a Help Balloon

Using the `HMExtractHelpMsg` and `HMGetIndHelpMsg` functions, you can extract information from existing help resources.

You can use `HMExtractHelpMsg` to extract the help messages specified in existing help resources. You might find this useful if you have duplicate commands and you want to store help messages in only one resource. For example, if you have a dialog box that replicates portions of a pull-down menu, you could specify help messages in the `'hmnu'` resource for the pull-down menu, and use `HMExtractHelpMsg` to extract those help messages to use with the related items in the dialog box's `'hdlg'` resource.

HMExtractHelpMsg

You can use the `HMExtractHelpMsg` function to extract the help balloon messages from existing help resources.

```
FUNCTION HMExtractHelpMsg (whichType: ResType;
                           whichResID, whichMsg,
                           whichState: Integer;
                           VAR aHelpMsg: HMMessageRecord): OSErr;
```

`whichType` The type of help resource. You can use one of these constants: `kHMMenuResType`, `kHMDialogResType`, `kHMRectListResType`, `kHMOverrideResType`, or `kHMFinderApplResType`.

`whichResID`
The resource ID of the help resource whose help message you wish to extract.

`whichMsg` The index of the component you wish to extract. The header and missing-items components don't count as components to index, because this function always skips those two components. For help resources that include both header and missing-items components, specify 1 to get the help messages contained in a help resource's menu-title component.

`whichState`
: For menu items and items in alert or dialog boxes, specifies the state of the item whose message you wish to extract. Use one of the following constants: `kHMEnabledItem`, `kHMDisabledItem`, `kHMCheckedItem`, or `kHMOtherItem`.

`aHelpMsg`
: A help message record.

DESCRIPTION

The `HMExtractHelpMsg` function returns in its `aHelpMsg` parameter the help message for an item in a specified state.

The `whichType` parameter identifies the type of resource from which you are extracting the help message. You can use one of these constants for the `whichType` parameter.

```
CONST kHMMenuResType       = 'hmnu';{menu help resource type}
      kHMDialogResType     = 'hdlg';{dialog help resource type}
      kHMWindListResType   = 'hwin';{window help resource type}
      kHMRectListResType   = 'hrct';{rectangle help resource type}
      kHMOverrideResType   = 'hovr';{help override resource }
                                    { type}
      kHMFinderApplResType = 'hfdr';{application icon help }
                                    { resource type}
```

The `whichState` parameter specifies the state of the item whose message you want to extract. You can use one of these constants for the `whichState` parameter.

```
CONST kHMEnabledItem  = 0; {enabled state for menu items; }
                           { contrlHilite value of 0 for }
                           { controls}
      kHMDisabledItem = 1; {disabled state for menu items; }
                           { contrlHilite value of 255 for }
                           { controls}
      kHMCheckedItem  = 2; {enabled-and-checked state for }
                           { menu items; contrlHilite value }
                           { of 1 for controls that are "on"}
      kHMOtherItem    = 3; {enabled-and-marked state for menu }
                           { items; contrlHilite value }
                           { between 2 and 253 for controls}
```

For the `kHMRectListResType`, `kHMOverrideResType`, and `kHMFinderApplResType` resource types—which don’t have states—supply the `kHMEnabledItem` constant for the `whichState` parameter.

The application-defined procedure shown in Listing 3-21 extracts the help balloon message from the 'hmnu' resource with a resource ID of 128. A value of 1 is supplied as the whichMsg parameter to retrieve information about the resource's first component (after the header and missing-items components, that is), which is the menu title. The menu title has four possible states; to retrieve the help message for the menu title in its dimmed state, the constant kHMDisabledItem is used for the whichState parameter. The help message record returned in aHelpMsg is then passed to HMShowBalloon, which displays the message in a balloon whose tip is located at the point specified in the tip parameter.

Listing 3-21 Using the HMExtractHelpMsg function

```
FUNCTION MyShowBalloonForDimMenuTitle: OSErr;
VAR
   aHelpMsg:         HMMessageRecord;
   tip:              Point;
   alternateRect:    Rect;
   err:              OSErr;
BEGIN
   err := HMExtractHelpMsg(kHMMenuResType, 128, 1,
                           kHMDisabledItem, aHelpMsg);
   IF err = noErr THEN
   {be sure to assign a tip and rectangle coordinates here}
      err := HMShowBalloon(aHelpMsg, tip, alternateRect,
                           NIL, 0, 0, kHMRegularWindow);
   MyShowBalloonForDimMenuTitle:= err;
END;
```

To retrieve all of the help messages for a given resource, set whichMsg to 1 and make repeated calls to HMExtractHelpMsg, incrementing whichMsg by 1 on each subsequent call until it returns the hmSkippedBalloon result code.

SPECIAL CONSIDERATIONS

If HMCompareItem appears as a component of an 'hmnu' resource that you're examining, neither this function nor HMGetIndHelpMsg performs a comparison against the current name of any menu item. Instead, these functions return the messages listed in your HMCompareItem components in the order in which they appear in the 'hmnu' resource.

When supplying an index for the whichMsg parameter, don't count the header component or the missing-items component as components to index. This function always skips both components; therefore, for help resources that include both header and missing-items components, specify 1 to get the help messages contained in a help resource's menu-title component.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMExtractHelpMsg` function are

Trap macro	Selector
`_Pack14`	$0711

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in parameter list
`memFullErr`	–108	Not enough room in heap zone
`resNotFound`	–192	Unable to read resource
`hmSkippedBalloon`	–857	No help message to fill in
`hmWrongVersion`	–858	Wrong version of Help Manager resource
`hmUnknownHelpType`	–859	Help message record contained a bad type

SEE ALSO

The `aHelpMsg` parameter is of data type `HMMessageRecord`. “Providing Help Balloons for Dynamic Windows” beginning on page 3-74 describes the fields of the help message record.

HMGetIndHelpMsg

To extract the help messages in existing help resources as well as additional information regarding the help resource, such as its variation code, tip location, and so on, use the `HMGetIndHelpMsg` function.

```
FUNCTION HMGetIndHelpMsg (whichType: ResType;
                          whichResID, whichMsg,
                          whichState: Integer;
                          VAR options: LongInt; VAR tip: Point;
                          VAR altRect: Rect; VAR theProc: Integer;
                          VAR variant: Integer;
                          VAR aHelpMsg: HMMessageRecord;
                          VAR count: Integer): OSErr;
```

`whichType` The type of help resource. You can use one of these constants: `kHMMenuResType`, `kHMDialogResType`, `kHMRectListResType`, `kHMOverrideResType`, or `kHMFinderApplResType`.

`whichResID`
The resource ID of the help resource whose help message you wish to extract.

`whichMsg`	The index of the component you wish to extract. The header and missing-items components don't count as components to index, because this function always skips those two components. For help resources that include both header and missing-items components, specify 1 to get the help messages contained in a help resource's menu-title component.
`whichState`	For menu items and items in alert and dialog boxes, specifies the state of the item whose message you wish to extract. Use one of the following constants: `kHMEnabledItem`, `kHMDisabledItem`, `kHMCheckedItem`, or `kHMOtherItem`.
`options`	The value of the `options` element of the help resource.
`tip`	The coordinates of the help balloon's tip location.
`altRect`	The coordinates of the help balloon's alternate rectangle.
`theProc`	The resource ID of the help balloon's `'WDEF'` resource.
`variant`	The balloon definition function's variation code.
`aHelpMsg`	The help message.
`count`	The number of components defined in the resource (not counting the header and missing-items components).

DESCRIPTION

Like the `HMExtractHelpMsg` function, the `HMGetIndHelpMsg` function returns in its `aHelpMsg` parameter the help message for an item in a specified state. The `HMGetIndHelpMsg` function uses additional parameters to return even more information about the help balloon than does `HMExtractHelpMsg`.

To retrieve all of the help balloon messages and related information for a given resource, set `whichMsg` to 1 and make repeated calls to `HMGetIndHelpMsg`, incrementing `whichMsg` by 1 on each subsequent call until it returns the `hmSkippedBalloon` result code.

The `whichType` parameter identifies the type of resource from which you are extracting the help message. You can use one of these constants for the `whichType` parameter.

```
CONST kHMMenuResType       = 'hmnu';{menu help resource type}
      kHMDialogResType     = 'hdlg';{dialog help resource type}
      kHMWindListResType   = 'hwin';{window help resource type}
      kHMRectListResType   = 'hrct';{rectangle help resource type}
      kHMOverrideResType   = 'hovr';{help override resource }
                                    { type}
      kHMFinderApplResType = 'hfdr';{application icon help }
                                    { resource type}
```

The `whichState` parameter specifies the state of the item whose message you want to extract. You can use one of these constants for the `whichState` parameter.

```
CONST kHMEnabledItem    = 0;  {enabled state for menu items; }
                              { contrlHilite value of 0 for }
                              { controls}
      kHMDisabledItem   = 1;  {disabled state for menu items; }
                              { contrlHilite value of 255 for }
                              { controls}
      kHMCheckedItem    = 2;  {enabled-and-checked state for }
                              { menu items; contrlHilite value }
                              { of 1 for controls that are "on"}
      kHMOtherItem      = 3;  {enabled-and-marked state for menu }
                              { items; contrlHilite value }
                              { between 2 and 253 for controls}
```

For the `kHMRectListResType`, `kHMOverrideResType`, and `kHMFinderApplResType` resource types—which don't have states—supply the `kHMEnabledItem` constant for the `whichState` parameter.

SPECIAL CONSIDERATIONS

If `HMCompareItem` appears as a component of an `'hmnu'` resource that you're examining, neither this function nor `HMExtractHelpMsg` performs a comparison against the current name of any menu item. Instead, these functions return the messages listed in your `HMCompareItem` components in the order in which they appear in the `'hmnu'` resource.

When supplying an index for the `whichMsg` parameter, don't count the header component or the missing-items component as components to index. This function always skips both components; therefore, for help resources that include both header and missing-items components, specify 1 to get the help messages contained in a help resource's menu-title component.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `HMGetIndHelpMsg` function are

Trap macro	Selector
`_Pack14`	$1306

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in parameter list
`memFullErr`	–108	Not enough room in heap zone
`resNotFound`	–192	Unable to read resource
`hmSkippedBalloon`	–857	No help message to fill in
`hmWrongVersion`	–858	Wrong version of Help Manager resource
`hmUnknownHelpType`	–859	Help message record contained a bad type

SEE ALSO

The `aHelpMsg` parameter is of data type `HMMessageRecord`. “Providing Help Balloons for Dynamic Windows” beginning on page 3-74 describes the fields of the help message record.

Application-Defined Routines

A balloon definition function is responsible for calculating the content region and structure region of the help balloon window and drawing the frame of the help balloon. The Help Manager takes care of positioning, sizing, and drawing your help balloons, and the standard balloon definition function provides a consistent and attractive shape to balloons across all applications. Though it takes extra work on your part, and your balloons will not share the consistent appearance of help balloons used by the Finder and by other applications, you can create your own balloon definition function, described in this section as `MyBalloonDef`.

When you use the `HMShowBalloon` and `HMShowMenuBalloon` functions to display help balloons, you pass a pointer to a tip function in the `tipProc` parameter. Normally, you supply `NIL` in this parameter to use the Help Manager’s default tip function. However, you can also supply your own tip function, described in this section as `MyTip`. The Help Manager calls your tip function after calculating the size and the location of a help balloon and before displaying it. This allows you to examine and, if necessary, adjust the balloon before it is displayed. For example, if you determine that the help balloon would obscure an object that requires extensive redrawing, you might use a different variation code to move the balloon.

MyBalloonDef

Here's a sample declaration for a balloon definition function called `MyBalloonDef`.

```
FUNCTION MyBalloonDef (variant: Integer; theBalloon: WindowPtr;
                       message: Integer;
                       param: LongInt): LongInt;
```

`variant` The variation code used to specify the shape and position of the help balloon. You should use the same relative position for the tip of the help balloon that the standard variation codes 0 through 7 specify. This ensures that the tip of the help balloon points to the object that the help balloon describes.

`theBalloon`
A pointer to the window of the help balloon.

`message` Identifies the action your balloon definition function should perform. Your balloon definition function can be sent the same messages as a window definition function, but the only ones your balloon definition function needs to process are the `wCalcRgns` and `wDraw` messages.

When your balloon definition function receives the `wCalcRgns` message, your function should calculate the content region and structure region of the help balloon.

When your balloon definition function receives the `wDraw` message, your function should draw the frame of the help balloon.

If you want to process other messages in your balloon definition function (for example, performing any additional initialization), you can also process the other standard `'WDEF'` messages.

`param` As with a window definition function, the value of this parameter depends on the value of the `message` parameter. Because this parameter is not used by the `wCalcRgns` and `wDraw` messages, your balloon definition function should disregard the value of this parameter.

DESCRIPTION

Your balloon definition function must define the appearance of the help balloon, which is a special type of window. You can implement your own balloon definition function by writing a window definition function that performs the tasks described in this section. (The standard balloon definition function is of type `'WDEF'` with resource ID 126.)

Your balloon definition function is also responsible for calculating the content region and structure region of the help balloon window and drawing the frame of the help balloon. The content region is the area inside the balloon frame; it contains the help message. The structure region is the boundary region of the entire balloon, including the content area and the pointer that extends from one of the help balloon's corners.

If you want the Help Manager to use your balloon definition function, you specify its resource ID and the desired variation code either in the `HMShowBalloon` function or in the appropriate elements of the `'hmnu'`, `'hdlg'`, or `'hrct'` resource. The Help Manager derives your balloon's window definition ID from its resource ID.

SEE ALSO

In the `variant` parameter, you should use the same relative position for the tip of the help balloon that the standard variation codes 0 through 7 specify, as illustrated in Figure 3-4 on page 3-10.

The `wCalcRgns` and `wDraw` messages are described in the chapter "Window Manager" of *Inside Macintosh: Macintosh Toolbox Essentials*.

MyTip

Here's a sample declaration of a tip function called `MyTip`.

```
FUNCTION MyTip (tip: Point; structure: RgnHandle; VAR r: Rect;
                VAR variant: Integer): OSErr;
```

`tip`	The location of the help balloon tip.
`structure`	A handle to the help balloon's region structure. The Help Manager returns this value. The structure region is the boundary region of the entire balloon, including the content area and the pointer that extends from one of the help balloon's corners.
`r`	The coordinates of the help balloon's content region. The content region is the area inside the balloon frame; it contains the help message. If this rectangle is not appropriate for the current screen display, you can specify different coordinates in this parameter.
`variant`	Variation code to be used for the help balloon. If this variation code is not appropriate for the current screen display, you can specify different coordinates in this parameter.

DESCRIPTION

Before displaying a help balloon created with the `HMShowBalloon` or `HMShowMenuBalloon` function, the Help Manager calls this function if you point to it in the `tipProc` parameter of either `HMShowBalloon` or `HMShowMenuBalloon`. The Help Manager returns the location of the help balloon tip, a handle to the help balloon's region structure, the coordinates of its content region, and the variation code to be used for the help balloon. If the help balloon that `HMShowBalloon` or `HMShowMenuBalloon` initially calculates is not appropriate for your current screen display, you can make minor adjustments to it by specifying a different rectangle in the `r` parameter (in which case the Help Manager automatically adjusts the `structure` parameter so that the entire balloon is larger or smaller as necessary) or by specifying a different variation code in the `variant` parameter.

If you need to make a major adjustment to the help balloon, return the `hmBalloonAborted` result code and call `HMShowBalloon` or `HMShowMenuBalloon` with appropriate new parameter values. To use the values returned in your tip function's parameters, return the `noErr` result code.

Listing 3-22 shows an example of using a tip function to refrain from displaying a balloon if it obscures an area of the screen that requires extensive drawing.

Listing 3-22 Using a tip function

```
VAR
   temprect:         Rect;
   DontObscureRect:  Rect;
   tip:              Point;
   structure:        RgnHandle;
   aHelpMsg:         HMMessageRecord;

BEGIN
      {be sure to determine DontObscureRect and fill in aHelpMsg}
      IF HMShowBalloon(aHelpMsg, tip, NIL, @MyTip, 0, 0,
                       kHMRegularwindow) = noErr
      THEN
         {test whether balloon obscures complex graphic }
         { in DontObscureRect}
         IF SectRect(structure^^.rgnBBox, DontObscureRect,
                     temprect) THEN
            {don't show this balloon but call HMShowBalloon later}
            MyTip := hmBalloonAborted
         ELSE  {use the balloon as calculated by the Help Manager}
            MyTip := noErr;
END;
```

SEE ALSO

Figure 3-4 on page 3-10 illustrates the structure regions and positions of the eight standard help balloons.

The `HMShowBalloon` function is described on page 3-100, and the `HMShowMenuBalloon` function is described on page 3-103.

Resources

This section describes the resources that the Help Manager uses to size, position, and draw help balloons for menus, alert and dialog boxes, static windows, non-document Finder icons, and several default help balloons provided by system software.

Help resources generally specify help messages, a balloon definition function, a variation code, and, when necessary, the balloon tip and either a hot rectangle or an alternate rectangle. The Help Manager uses this information as appropriate when drawing help balloons. These help resources are

- the menu help (`'hmnu'`) resource, which provides help balloons for menus and menu items
- the dialog-item help (`'hdlg'`) resource, which provides help balloons for items in dialog boxes and alert boxes
- the rectangle help (`'hrct'`) resource, which associates a help balloon with a hot rectangle in a static window
- the window help (`'hwin'`) resource, which associates an `'hrct'` or `'hdlg'` resource with a hot rectangle in a window or with an item in a dialog box or alert box
- the Finder icon help (`'hfdr'`) resource, which provides a custom help balloon for your application icon
- the default help override (`'hovr'`) resource, which overrides the help messages of default help balloons provided in system software

This section describes the structures of these resources after they are compiled by the Rez resource compiler, available from APDA. If you are interested in creating the Rez input files for these resources, see “Using the Help Manager” beginning on page 3-18 for detailed information.

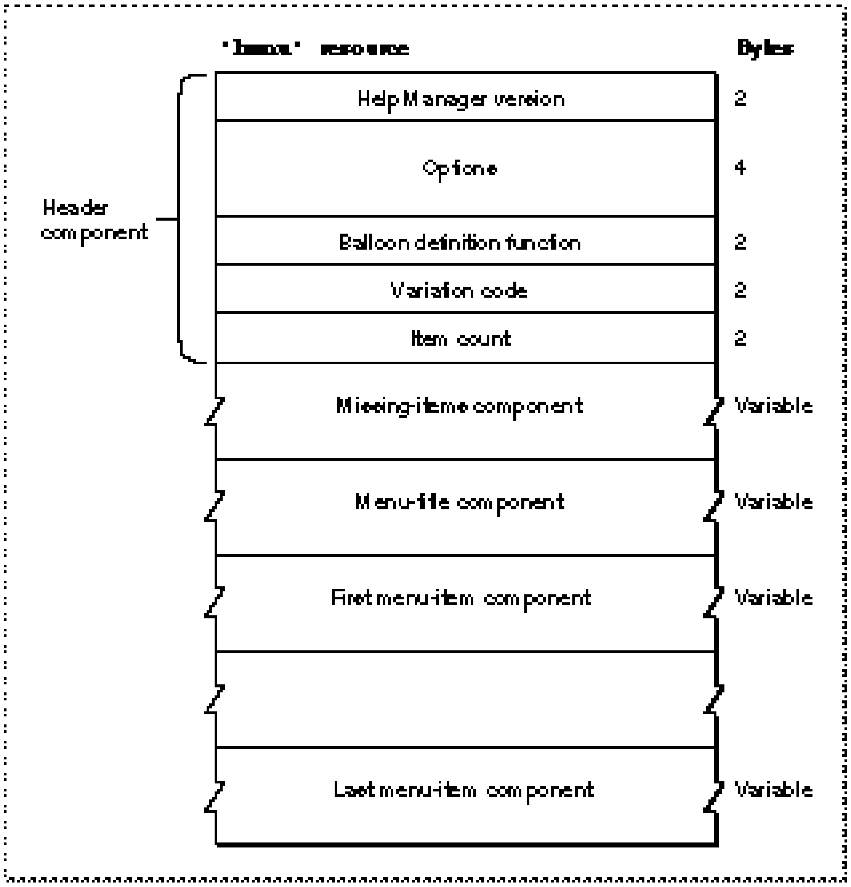
The Menu Help Resource

To provide help balloons for a menu—pull-down, pop-up, or hierarchical—that uses the standard menu definition procedure, you can create a menu help resource. A menu help resource is a resource of type `'hmnu'`; in it, you specify help balloons for the menu title and for each item in the menu. You create a separate `'hmnu'` resource for each menu. All `'hmnu'` resources must have resource IDs greater than 128.

The format of a Rez input file for an `'hmnu'` resource differs from its compiled output form. This section describes the structure of a Rez-compiled `'hmnu'` resource. If you are concerned only with creating `'hmnu'` resources, see “Providing Help Balloons for Menus” beginning on page 3-27. That section gives a detailed description, using several code samples, of how to use Rez input files to create `'hmnu'` resources.

An 'hmnu' resource consists of a header component, a missing-items component, a menu-title component, and a variable number of menu-item components. Figure 3-23 shows the general structure of a compiled 'hmnu' resource.

Figure 3-23 Structure of a compiled menu help ('hmnu') resource



If you examine a compiled version of an 'hmnu' resource, you find that the header component consists of the following elements:

- Help Manager version. The version of the Help Manager to use; specified in a Rez input file with the HelpMgrVersion constant.
- Options. The sum of the values of available options, described in “Specifying Options in Help Resources” beginning on page 3-25.
- Balloon definition function. The resource ID of the window definition function used for drawing the help balloon. The standard balloon definition function is of type 'WDEF' with resource ID 126; this can be specified by the number 0 in the Rez input file.

- Variation code. A number signifying the preferred position of the help balloon relative to the hot rectangle. The balloon definition function draws the frame of the help balloon based on the variation code specified here. The eight variation codes and how they affect the standard balloon definition function are illustrated in Figure 3-4 on page 3-10.
- Item count. The number of remaining components—including the missing-items, menu-title, and menu-item components—defined in the rest of this resource.

The Help Manager identifies each component by its order in the resource. The missing-items component always follows the header component of an `'hmnu'` resource. The menu-title component always follows the missing-items component. Then a variable number of menu-item components are stored in this resource. The Help Manager determines the end of the `'hmnu'` resource by using the item count information in the header component.

The structures of the missing-items component, the menu-title component, and the menu-item components depend on identifiers specified inside the components. The identifiers used in a Rez input file are described in “Specifying the Format for Help Messages” on page 3-23.

The missing-items component, the menu-title component, and the menu-item components can each specify four different help messages:

- First help message.
  - In the missing-items component, this is the help message for missing enabled items.
  - In the menu-title component, this is the help message for the enabled menu title.
  - In all subsequent menu-item components, this is the help message for enabled menu items.
- Second help message.
  - In the missing-items component, this is the help message for missing items that are dimmed by the application.
  - In the menu-title component, this is the help message for the menu title when the application dims it.
  - In all subsequent menu-item components, this is the help message for menu items when the application dims them.
- Third help message.
  - In the missing-items component, this is the help message for missing enabled-and-checked items.
  - In the menu-title component, this is the help message for the menu title when system software dims it at the appearance of an alert box or a modal dialog box.
  - In all subsequent menu-item components, this is the help message for enabled-and-checked menu items.

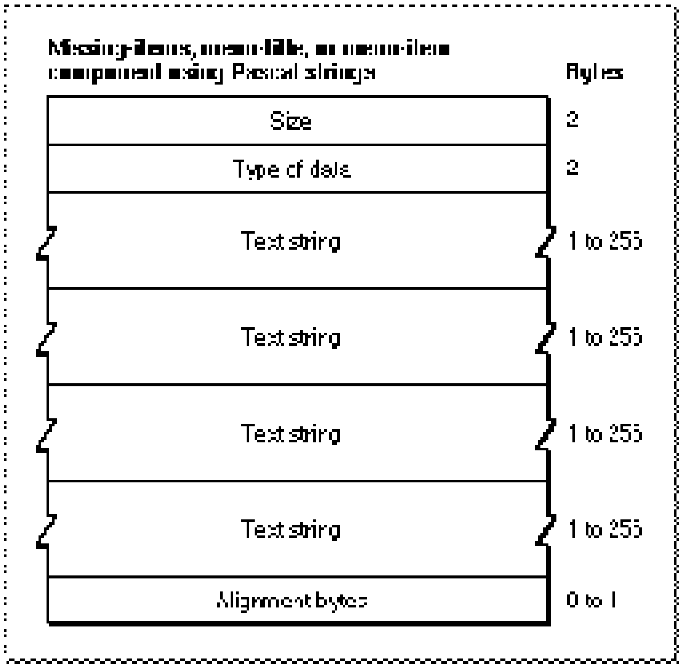
- Fourth help message.
  - In the missing-items component, this is the help message for missing enabled-and-marked items.
  - In the menu-title component, this is the help message for all menu items when system software dims them at the appearance of an alert box or a modal dialog box.
  - In all subsequent menu-item components, this is the help message for enabled-and-marked menu items.

An empty string or a resource ID of 0 for any messages in the menu-title or menu-item components causes the Help Manager to use the appropriate help message contained in the missing-items component.

Since they all adhere to the formats specified by the previously described identifiers, the missing-items component, the menu-title component, and the menu-item components can have similar structures. The Help Manager determines the end of a component by examining its length, which is stored in the first 2 bytes of the component.

Figure 3-24 shows the structure of a component that stores its help messages as Pascal strings within the `'hmnu'` resource itself.

Figure 3-24 Structure of an `'hmnu'` component compiled with the `HMStringItem` identifier

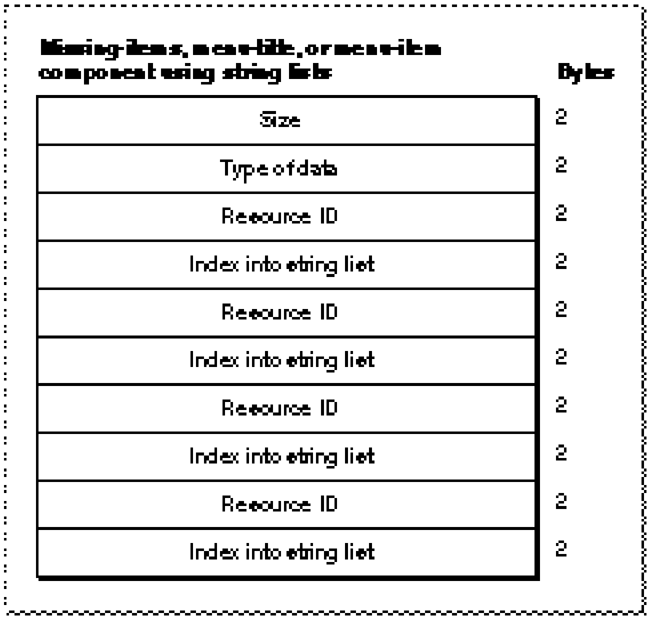


If you examine a compiled version of an `'hmnu'` resource, you find that a component identified in a Rez input file by the `HMStringItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data. The value 1 is specified here when the help messages are stored as Pascal strings within this component.
- Text string. The first help message (as previously described).
- Text string. The second help message (as previously described).
- Text string. The third help message (as previously described).
- Text string. The fourth help message (as previously described).
- Alignment bytes. Zero or one bytes used to make the previous text strings end on a word boundary.

Figure 3-25 shows the structure of an `'hmnu'` component that specifies its help messages as text strings stored in string list (`'STR#'`) resources.

Figure 3-25 Structure of an `'hmnu'` component compiled with the `HMStringResItem` identifier



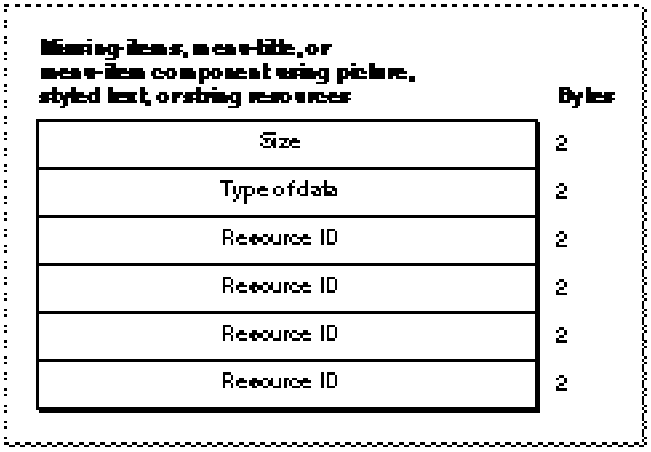
If you examine a compiled version of an `'hmnu'` resource, you find that a component identified in a Rez input file by the `HMStringResItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data. The value 3 is specified here when the help messages for this component are stored in string list (`'STR#'`) resources.
- Resource ID. The resource ID of an `'STR#'` resource.
- Index into the string list resource. A number used as an index to a particular text string within the `'STR#'` resource. This text string is used for the first help message (as previously described).

Three more pairs of resource IDs/index numbers follow. The text strings that these pairs refer to are used for the second, third, and fourth help messages, respectively.

Figure 3-26 shows the structure of an `'hmnu'` component that specifies its help messages in picture (`'PICT'`) resources, styled text (`'TEXT'` and `'styl'`) resources, or string (`'STR '`) resources.

Figure 3-26 Structure of an `'hmnu'` component compiled with the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier


Missing-items, menu-title, or
menu-item component using picture,
styled text, or string resources
Bytes
Size
2
Type of data
2
Resource ID
2
Resource ID
2
Resource ID
2
Resource ID
2

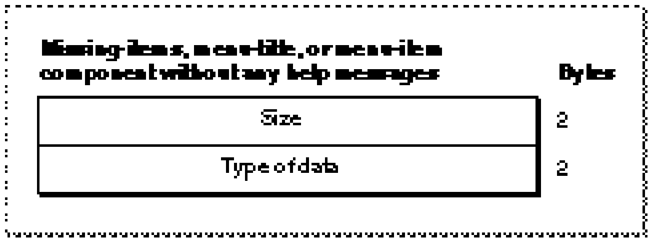
If you examine a compiled version of an `'hmnu'` resource, you find that a component identified in a Rez input file by either the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data.
  - The value 2 is specified here when the help messages for this component are stored in `'PICT'` resources.
  - The value 6 is specified here when the help messages for this component are stored as styled text—that is, in both `'TEXT'` and `'styl'` resources.
  - The value 7 is specified here when the help messages for this component are stored in `'STR '` resources.
- Resource ID.
  - The resource ID of a `'PICT'` resource when the value 2 is specified as the type of data. The Help Manager uses the picture contained in this resource for the first help message (as previously described).
  - The resource ID common to both a `'TEXT'` and an `'styl'` resource when the value 6 is specified as the type of data. The Help Manager uses the styled text specified by these resources for the first help message.
  - The resource ID of an `'STR '` resource when the value 7 is specified as the type of data. The Help Manager uses the text contained in this resource for the first help message.

Three more resource IDs follow; the Help Manager uses these resources (either `'PICT'`, `'TEXT'` and `'styl'`, or `'STR '`) for the second, third, and fourth help messages, respectively (as previously described).

Figure 3-27 shows the structure of an `'hmnu'` component that specifies no help messages.

Figure 3-27 Structure of an `'hmnu'` component compiled with the `HMSkipItem` identifier

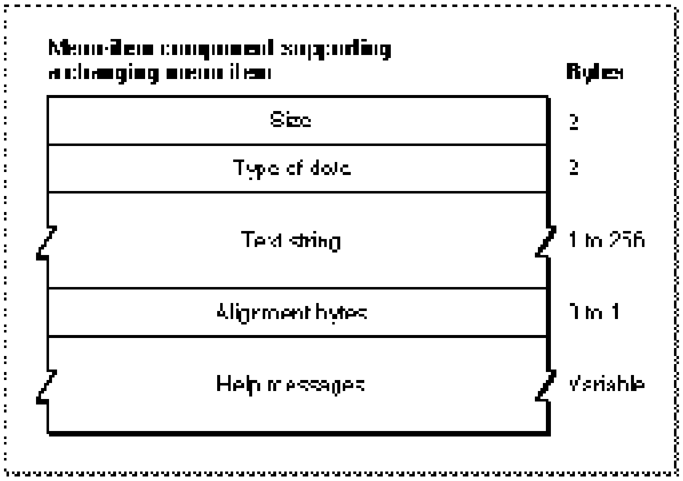

Missing-items, menu-title, or menu-item component without any help messages
Bytes
Size
2
Type of data
2

If you examine a compiled version of an `'hmnu'` resource, you find that a component identified by the `HMSkipItem` identifier consists of the following elements:

- Size. The value 4, for the number of bytes contained in this component.
- Type of data. The value 256.

For menu-item components, two additional identifiers are available: HMCompareItem and HMNamedResourceItem. When the HMCompareItem identifier is specified, the Help Manager compares a string specified in the component against the current menu item. If the string matches the current menu item, the Help Manager uses the help messages specified in the rest of the component, shown in Figure 3-28. This type of component is useful for a menu item that can change names.

Figure 3-28 Structure of a menu-item component compiled with the HMCompareItem identifier



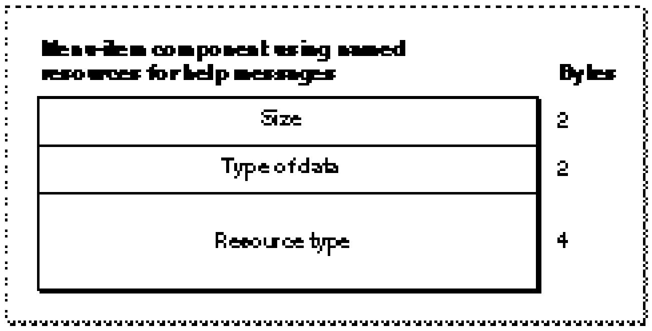
If you examine a compiled version of an 'hmnu' resource, you find that a component identified in a Rez input file by the HMCompareItem identifier consists of these elements:

- Size. The number of bytes contained in this component.
- Type of data. The value 512 appears here when the Help Manager is to use the help messages specified in this component only when the current menu item matches a specified text string.
- Text string. The string against which to compare the current menu item. If the current menu item matches this string, then the Help Manager uses the help messages specified in this component.
- Alignment bytes. Zero or one bytes used to make the previous text string end on a word boundary.
- Help messages. The four help messages for the menu item. The structure may follow that of any of the previously described menu-item components; that is, this element consists of a value representing the format of the help messages specified in the rest of the component, the size of the rest of the component, and specifications for four actual help messages for the menu item.

When the identifier HMNamedResourceItem is specified, the Help Manager retrieves help messages from a resource that matches the name and state of the current menu item.

Figure 3-29 shows the format of a menu-item component that uses named resources for help messages.

Figure 3-29 Structure of a menu-item component compiled with the `HMNamedResourceItem` identifier



If you examine a compiled version of an `'hmnu'` resource, you find that a component identified in a Rez input file by the `HMNamedResourceItem` identifier consists of these elements:

- Size. The number of bytes contained in this component.
- Type of data. The number 1024 is specified here when the Help Manager is to use named resources for help messages.
- Resource type. The resource type (`'STR '`, `'STR#'`, `'PICT'`, or, for text, `'TEXT'`) of the resource that contains the help messages for the current menu item. The Help Manager then uses the `GetNamedResource` function to find the resource with the same name as the current menu item. (If `'TEXT'` is specified, the Help Manager also uses the style information contained in an `'styl'` resource with the same name.) If the menu item is dimmed, the Help Manager appends an exclamation point (!) to the menu item string and searches for a resource by that name. If the menu item is enabled and marked with a checkmark or other mark, the Help Manager appends the mark to the menu item string and looks for a resource with that name.

The Dialog-Item Help Resource

You can provide help balloons for individual items in a dialog box or an alert box by supplying a dialog-item help resource, which is a resource of type `'hdlg'`. You specify different help balloons for various states of an item—by highlight value if the item is a control, and by enabled or disabled states for items that are not controls.

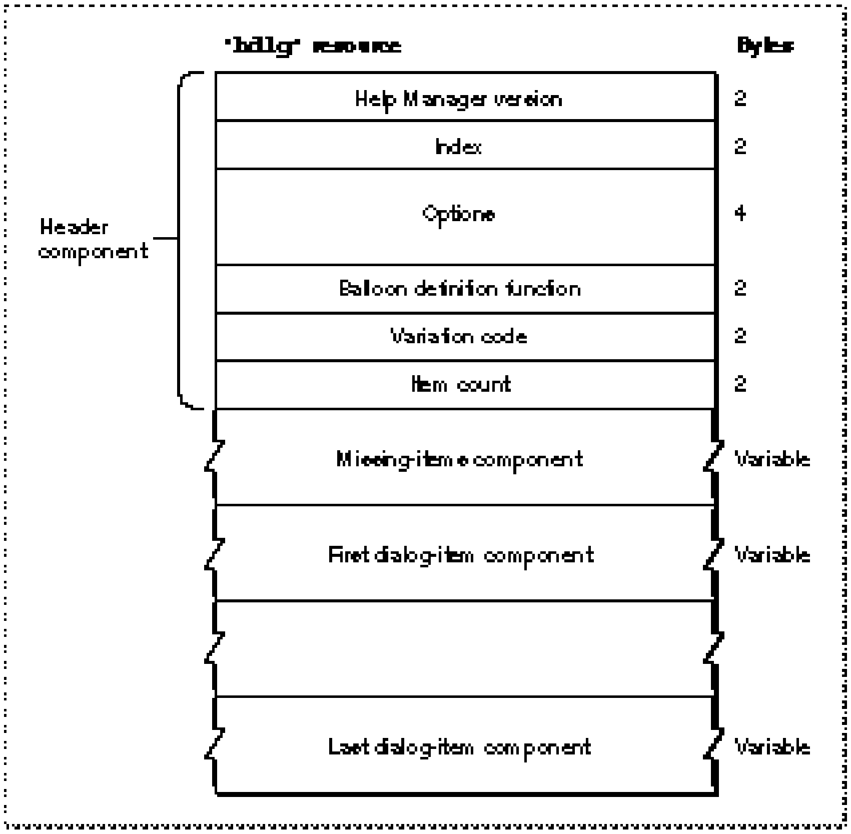
To associate an `'hdlg'` resource with a particular alert box or dialog box, either you must include an item of type `HelpItem` in the box's item list (`'DITL'`) resource, or you must create an `'hwin'` resource. Listing 3-8 on page 3-59 shows how to use an item of type `HelpItem`—and Listing 3-10 on page 3-72 shows you how to use an `'hwin'` resource—for associating an `'hdlg'` resource with a particular alert box or dialog box. For detailed information about using an item of type `HelpItem`, see "Using a Help Item Versus Using an 'hwin' Resource" on page 3-63. For detailed information on using an `'hwin'` resource, see "Associating Help Resources With Static Windows" on page 3-68.

All `'hdlg'` resources must have resource IDs greater than 128.

The format of a Rez input file for an `'hdlg'` resource differs from its compiled output form. This section describes the structure of a Rez-compiled `'hdlg'` resource. If you are concerned only with creating `'hdlg'` resources, see "Providing Help Balloons for Items in Dialog Boxes and Alert Boxes" on page 3-51 for a detailed description, using several code samples, of how to use Rez input files to create `'hdlg'` resources.

An `'hdlg'` resource consists of a header component, a missing-items component, and a variable number of dialog-item components. Figure 3-30 shows the general structure of a compiled `'hdlg'` resource.

Figure 3-30 Structure of a compiled dialog-item help (`'hdlg'`) resource



If you examine a compiled version of an `'hdlg'` resource, you find that the header component consists of the following elements:

- Help Manager version. The version of the Help Manager to use. This is usually specified in a Rez input file with the `HelpMgrVersion` constant.
- Index. An index (starting with 0) into an item list (`'DITL'`) resource. The Help Manager adds the value of this index to the number of the first item in the item list resource and then associates the result with an item number within the item list resource; therefore, index 0 corresponds to item 1 in the item list resource (because 0 plus 1 equals 1). The Help Manager then uses the first dialog-item component in the `'hdlg'` resource to provide help for the item to which this index corresponds. Subsequent dialog-item components specify help messages for subsequent items in the item list resource. For example, when 4 is specified as the index, the first dialog-item component specifies help messages for the fifth item in an item list resource. (As explained earlier, either an item of type `helpItem` in the item list resource or an `'hwin'` resource is used to associate the messages in the dialog-item components of this `'hdlg'` resource with the items of a particular dialog box or alert box.)
- Options. The sum of the values of available options, described in “Specifying Options in Help Resources” beginning on page 3-25.
- Balloon definition function. The resource ID of the window definition function used for drawing the help balloon. The standard balloon definition function is of type `'WDEF'` with resource ID 126; this can be specified by the number 0 in the Rez input file.
- Variation code. A number signifying the preferred position of the help balloon relative to the hot rectangle. The balloon definition function draws the frame of the help balloon based on the variation code specified here. The eight variation codes and how they affect the standard balloon definition function are illustrated in Figure 3-4 on page 3-10.
- Item count. The number of remaining components—that is, the missing-items component plus all dialog-item components—defined in the rest of this resource.

The missing-items component always follows the header component of an `'hdlg'` resource. Then a variable number of dialog-item components are stored in this resource. The Help Manager determines the end of the `'hdlg'` resource by using the item count information in the header component. The Help Manager determines the type of each component by its order in the resource.

The structures of the missing-items component and the dialog-item components depend on identifiers specified inside the components. The identifiers used in a Rez input file are described in “Specifying the Format for Help Messages” on page 3-23.

The missing-items component and the dialog-item components can each specify four different help messages:

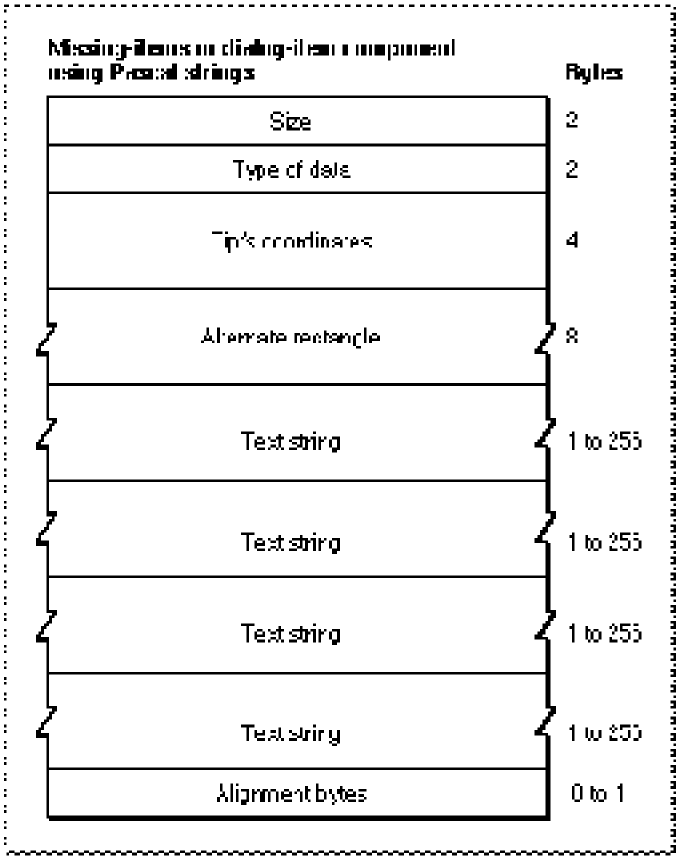
- First help message.
    - In the missing-items component, this is the help message both for missing, active, unselected controls (that is, those with highlight values of 0) and for missing enabled items that are not controls.
    - In dialog-item components, this is the help message for an active, unselected control (that is, one with a highlight value of 0) or for an enabled item that is not a control.
- Second help message.
    - In the missing-items component, this is the help message both for missing dimmed controls (that is, those with highlight values of 255) and for missing disabled items that are not controls.
    - In dialog-item components, this is the help message for a dimmed control (that is, one with a highlight value of 255) or for a disabled item that is not a control.
- Third help message.
    - In the missing-items component, this is the help message for missing active controls that are checked (that is, those with highlight values of 1).
    - In dialog-item components, this is the help message for an active control that is checked (that is, one with a highlight value of 1).
- Fourth help message.
    - In the missing-items component, this is the help message for missing, selected controls with highlight values between 2 and 253.
    - In dialog-item components, this is the help message for a selected control with any highlight value between 2 and 253.

An empty string or a resource ID of 0 for a message in any dialog-item component causes the Help Manager to use the appropriate help message contained in the missing-items component.

Since they both adhere to the formats specified by the previously described identifiers, the missing-items component and the dialog-item components can have similar structures. The Help Manager determines the end of a component by examining its length, which is stored in the first 2 bytes of the component.

Figure 3-31 shows the structure of a component that stores its help messages as Pascal strings within the `'hdlg'` resource itself.

Figure 3-31 Structure of an `'hdlg'` component compiled with the `HMStringItem` identifier



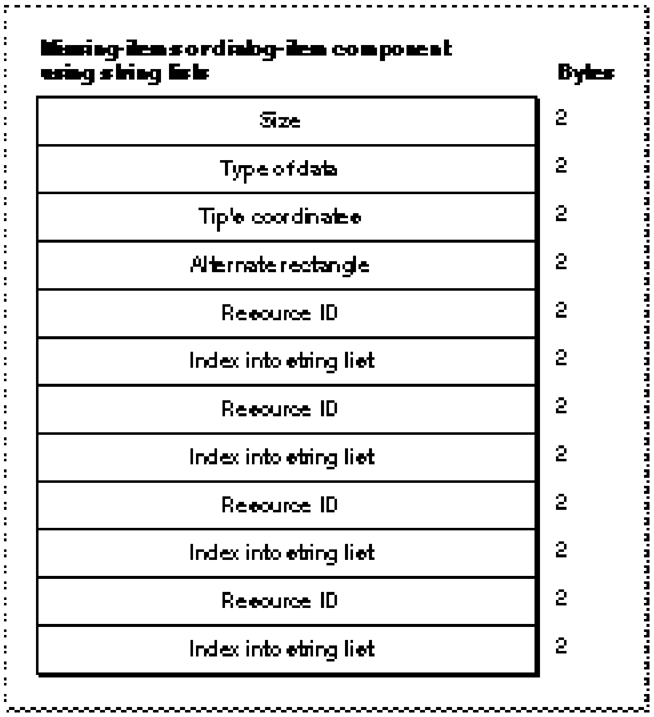
If you examine a compiled version of an `'hdlg'` resource, you find that a component identified in a Rez input file by the `HMStringItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data. The value 1 is specified here when the help messages are stored as Pascal strings within this component.
- Tip's coordinates. The coordinates of the help balloon's tip. The tip's coordinates are local to the item's display rectangle.
- Alternate rectangle. The coordinates for a rectangle used by the Help Manager for transposing the tip if a help balloon does not fit onscreen. These coordinates are local to the item's display rectangle.

- Text string. The first help message (as previously described).
- Text string. The second help message (as previously described).
- Text string. The third help message (as previously described).
- Text string. The fourth help message (as previously described).
- Alignment bytes. Zero or one bytes used to make the previous text strings end on a word boundary.

Figure 3-32 shows the structure of an `'hdlg'` component that specifies its help messages as text strings stored in string list (`'STR#'`) resources.

Figure 3-32 Structure of an `'hdlg'` component compiled with the `HMStringResItem` identifier



If you examine a compiled version of an `'hdlg'` resource, you find that a component identified in a Rez input file by the `HMStringResItem` identifier consists of the following elements:

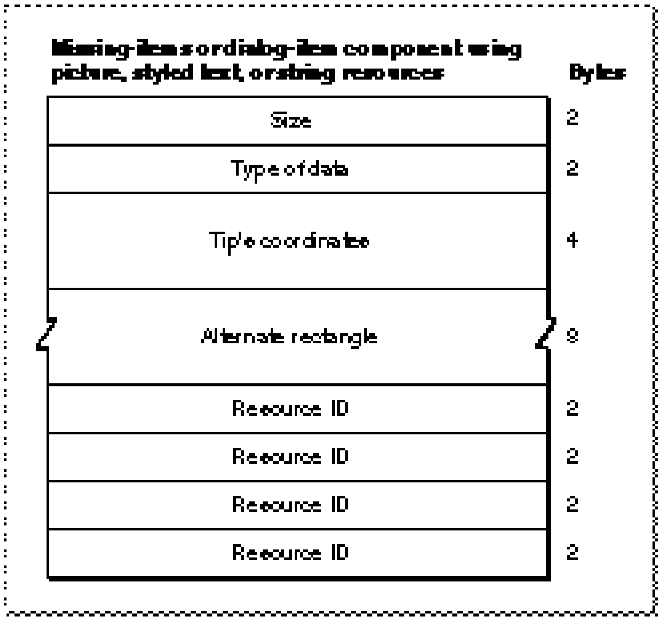
- Size. The number of bytes contained in this component.
- Type of data. The value 3 is specified here when the help messages for this component are stored in string list (`'STR#'`) resources.
- Tip's coordinates. The coordinates of the help balloon's tip. The tip's coordinates are local to the item's display rectangle.

- Alternate rectangle. The coordinates for a rectangle used by the Help Manager for transposing the tip if a help balloon does not fit onscreen. These coordinates are local to the item's display rectangle.
- Resource ID. The resource ID of an `'STR#'` resource.
- Index into the string list resource. A number used as an index to a particular text string within the `'STR#'` resource. This text string is used for the first help message (as previously described).

Three more pairs of resource IDs and their index numbers follow. The text strings referenced by these pairs are used for the second, third, and fourth help messages, respectively.

Figure 3-33 shows the structure of an `'hdlg'` component that specifies its help messages in picture (`'PICT'`) resources, styled text (`'TEXT'` and `'styl'`) resources, or string (`'STR '`) resources.

Figure 3-33 Structure of an `'hdlg'` component compiled with the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier



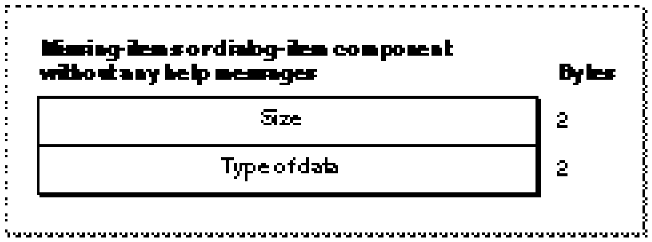
If you examine a compiled version of an `'hdlg'` resource, you find that a component identified in a Rez input file by either the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data.
  - The value 2 is specified here when the help messages for this component are stored in `'PICT'` resources.
  - The value 6 is specified here when the help messages for this component are stored as styled text—that is, in both `'TEXT'` and `'styl'` resources.
  - The value 7 is specified here when the help messages for this component are stored in `'STR '` resources.
- Tip's coordinates. The coordinates of the help balloon's tip. The tip's coordinates are local to the item's display rectangle.
- Alternate rectangle. The coordinates for a rectangle used by the Help Manager for transposing the tip if a help balloon does not fit onscreen. These coordinates are local to the item's display rectangle.
- Resource ID.
  - The resource ID of a `'PICT'` resource when the value 2 is specified as the type of data. The Help Manager uses the picture contained in this resource for the first help message (as previously described).
  - The resource ID common to both a `'TEXT'` and an `'styl'` resource when the value 6 is specified as the type of data. The Help Manager uses the styled text specified by these resources for the first help message.
  - The resource ID of an `'STR '` resource when the value 7 is specified as the type of data. The Help Manager uses the text contained in this resource for the first help message.

Three more resource IDs follow; the Help Manager uses these resources (either `'PICT'`, `'TEXT'` and `'styl'`, or `'STR '`) for the second, third, and fourth help messages, respectively (as previously described).

Figure 3-34 shows the structure of an `'hdlg'` component that specifies no help messages.

Figure 3-34 Structure of an `'hdlg'` component compiled with the `HMSkipItem` identifier


Missing-items or dialog-item component without any help messages
Bytes
Size
2
Type of data
2

If you examine a compiled version of an `'hdlg'` resource, you find that a component identified by the `HMSkipItem` identifier consists of the following elements:

- Size. The value 4, for the number of bytes contained in this component.
- Type of data. The value 256.

The Rectangle Help Resource

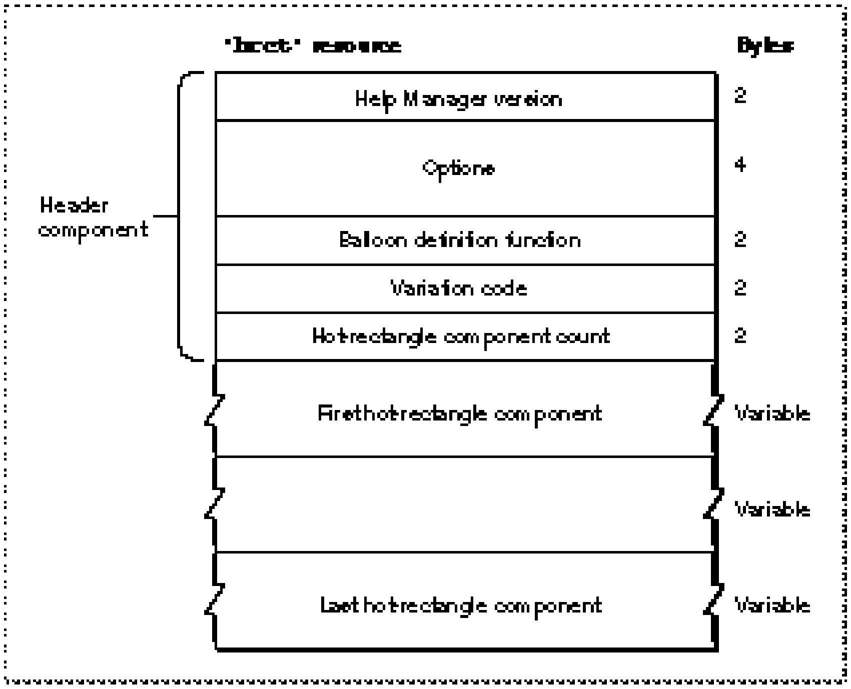
You can use a rectangle help resource to define hot rectangles for displaying help balloons within a static window, and to specify the help messages for those balloons. A rectangle help resource is a resource of type `'hrct'`. All `'hrct'` resources must have resource IDs greater than 128.

To associate the hot rectangles and help messages defined in an `'hrct'` resource with a particular window, you must also create a window help (`'hwin'`) resource, which is described in "Associating Help Resources With Static Windows" on page 3-68.

The format of a Rez input file for an `'hrct'` resource differs from its compiled output form. This section describes the structure of a Rez-compiled `'hrct'` resource. If you are concerned only with creating `'hrct'` resources, see "Specifying Help for Rectangles in Windows" on page 3-67 for a detailed description of how to use Rez input files to create `'hrct'` resources.

An `'hrct'` resource consists of a header component and a variable number of hot-rectangle components. Figure 3-35 shows the general structure of a compiled `'hrct'` resource.

Figure 3-35 Structure of a compiled rectangle help (`'hrct'`) resource


'hrct' resource
Bytes
Header component
Help Manager version
2
Options
4
Balloon definition function
2
Variation code
2
Hot-rectangle component count
2
First hot-rectangle component
Variable
Variable
Last hot-rectangle component
Variable

If you examine a compiled version of an `'hrct'` resource, you find that the header component consists of the following elements:

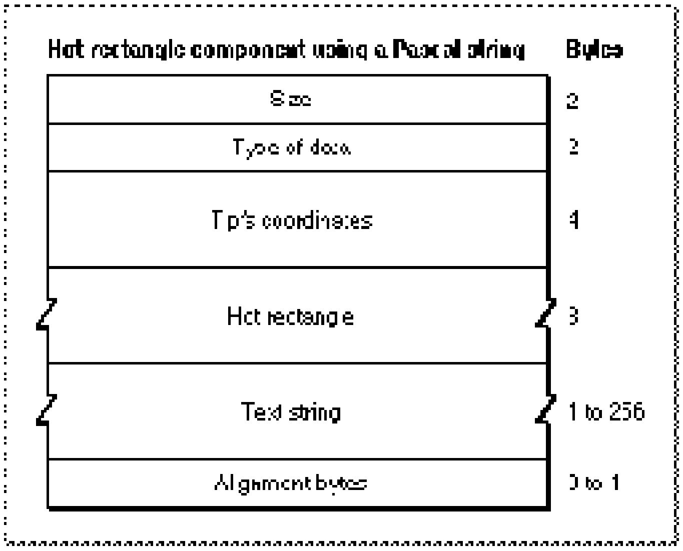
- Help Manager version. The version of the Help Manager to use. This is usually specified in a Rez input file with the `HelpMgrVersion` constant.
- Options. The sum of the values of available options, described in "Specifying Options in Help Resources" beginning on page 3-25.
- Balloon definition function. The resource ID of the window definition function used for drawing the help balloon. The standard balloon definition function is of type `'WDEF'` with resource ID 126; this can be specified by 0 in the Rez input file.
- Variation code. A number signifying the preferred position of the help balloon relative to the hot rectangle. The balloon definition function draws the frame of the help balloon based on the variation code specified here. The eight variation codes and how they affect the standard balloon definition function are illustrated in Figure 3-4 on page 3-10.
- Hot-rectangle component count. The number of hot-rectangle components defined in the rest of this resource.

The Help Manager determines the end of the `'hrct'` resource by using the component count information in the header component.

The structures of the hot-rectangle components depend on identifiers specified inside the components. The identifiers used in a Rez input file are described in “Specifying the Format for Help Messages” on page 3-23.

Figure 3-36 shows the structure of a component that stores its help message as a Pascal string within the `'hrct'` resource itself.

Figure 3-36 Structure of an `'hrct'` component compiled with the `HMStringItem` identifier



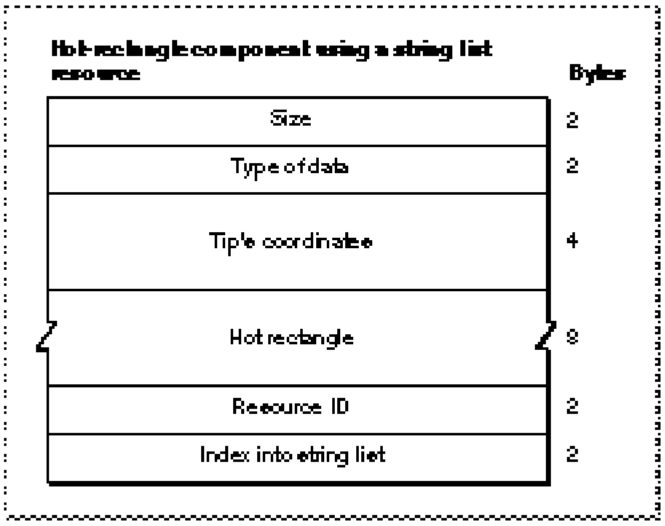
If you examine a compiled version of an `'hrct'` resource, you find that a component identified in a Rez input file by the `HMStringItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data. The value 1 is specified here when the help message is stored as a Pascal string within this component.
- Tip’s coordinates. The coordinates of the help balloon’s tip. The tip’s coordinates are local to the window.
- Hot rectangle. The coordinates (local to the window) of a rectangle. The Help Manager displays a help message when the user moves the cursor over this rectangle.

- Text string. The help message that the Help Manager displays when the user moves the cursor over the hot rectangle.
- Alignment bytes. Zero or one bytes used to make the previous text strings end on a word boundary.

Figure 3-37 shows the structure of a hot-rectangle component that specifies its help message as a text string stored in a string list (`'STR#'`) resource.

Figure 3-37 Structure of an `'hrct'` component compiled with the `HMStringResItem` identifier

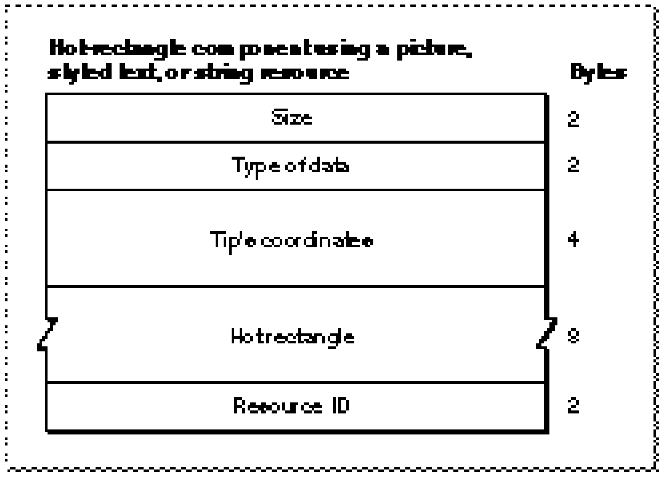


If you examine a compiled version of an `'hrct'` resource, you find that a component identified in a Rez input file by the `HMStringResItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data. The value 3 is specified here when the help message for this component is stored in an `'STR#'` resource.
- Tip's coordinates. The coordinates of the help balloon's tip. The tip's coordinates are local to the window.
- Hot rectangle. The coordinates (local to the window) of a rectangle. The Help Manager displays a help message when the user moves the cursor over this rectangle.
- Resource ID. The resource ID of an `'STR#'` resource.
- Index into the string list resource. A number used as an index to a particular text string within the `'STR#'` resource. When the user moves the cursor over the hot rectangle, the Help Manager displays this text string for the help message.

Figure 3-38 shows the structure of a hot-rectangle component that specifies its help message in a picture (`'PICT'`) resource, in styled text (`'TEXT'` and `'styl'`) resources, or in a string (`'STR '`) resource.

Figure 3-38 Structure of an `'hrct'` component compiled with the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier


Hot-rectangle component using a picture, styled text, or string resource
Bytes
Size
2
Type of data
2
Tip's coordinates
4
Hot rectangle
8
Resource ID
2

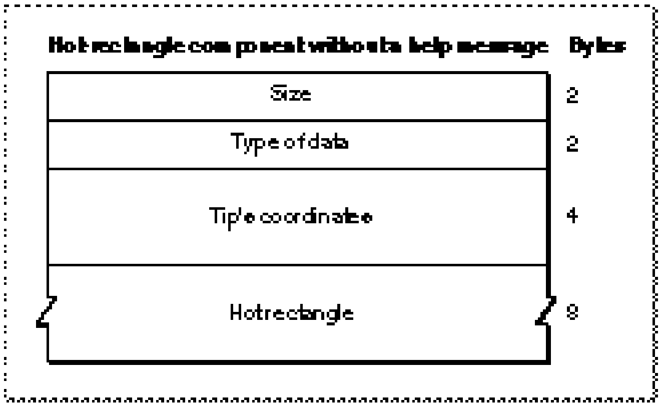
If you examine a compiled version of an `'hrct'` resource, you find that a component identified in a Rez input file by either the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data.
  - The value 2 is specified here when the help message for this component is stored in a `'PICT'` resource.
  - The value 6 is specified here when the help message for this component is stored as styled text—that is, in both `'TEXT'` and `'styl'` resources.
  - The value 7 is specified here when the help message for this component is stored in an `'STR '` resource.
- Tip's coordinates. The coordinates of the help balloon's tip. The tip's coordinates are local to the window.
- Hot rectangle. The coordinates (local to the window) of a rectangle. The Help Manager displays a help message when the user moves the cursor over this rectangle.

- Resource ID.
  - The resource ID of a `'PICT'` resource when the value 2 is specified as the type of data. When the user moves the cursor over the hot rectangle, the Help Manager displays the picture stored in this resource for the help message.
  - The resource ID common to both a `'TEXT'` and an `'styl'` resource when the value 6 is specified as the type of data. When the user moves the cursor over the hot rectangle, the Help Manager displays the styled text specified in these resources for the help message.
  - The resource ID of an `'STR '` resource when the value 7 is specified as the type of data. When the user moves the cursor over the hot rectangle, the Help Manager uses the text string stored in this resource for the help message.

Figure 3-39 shows the structure of a hot-rectangle component that doesn't specify a help message.

Figure 3-39 Structure of an `'hrct'` component compiled with the `HMSkipItem` identifier



If you examine a compiled version of an `'hrct'` resource, you find that a component identified by the `HMSkipItem` identifier consists of the following elements:

- Size. The value 4, for the number of bytes contained in this component.
- Type of data. The value 256.
- Tip's coordinates. In this instance, the Help Manager does not use this information because it does not display a help balloon.
- Hot rectangle. The coordinates (local to the window) of a rectangle that is to be skipped. When the user moves the cursor over this rectangle, the Help Manager does *not* display any help messages.

The Window Help Resource

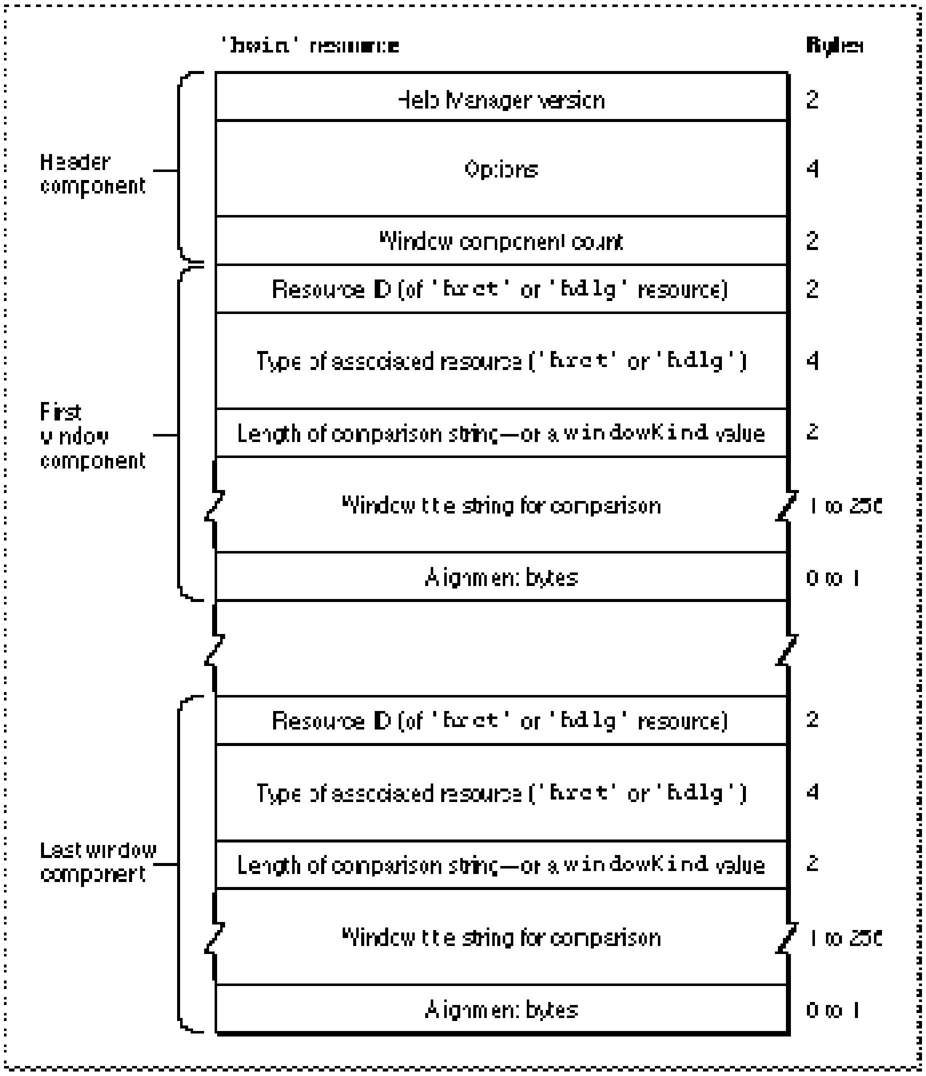
To associate the help balloons defined in an 'hrct' resource with a particular window, you must create a window help resource. Unless you include an item of type HelpItem in an item list resource, you also must create a window help resource to associate an 'hdlg' resource with a particular alert box or dialog box. The window help resource is a resource of type 'hwin'. All 'hwin' resources must have resource IDs greater than 128.

The 'hwin' resource merely associates 'hrct' and 'hdlg' resources with windows. To specify hot rectangles, help balloon characteristics, and help messages for areas in a static window, you must use 'hrct' or 'hdlg' resources, which are described in "Specifying Help for Rectangles in Windows" on page 3-67 and "Providing Help Balloons for Items in Dialog Boxes and Alert Boxes" on page 3-51, respectively.

The format of a Rez input file for an 'hwin' resource differs from its compiled output form. This section describes the structure of a Rez-compiled 'hwin' resource. If you are concerned only with creating 'hwin' resources, see "Associating Help Resources With Static Windows" on page 3-68 for a detailed description of how to use Rez input files to create 'hwin' resources.

An 'hwin' resource consists of a header component and a variable number of window components. Figure 3-40 shows the general structure of a compiled 'hwin' resource.

Figure 3-40 Structure of a compiled window help (`'hwin'`) resource


'hwin' resource
Bytes
Header component
Help Manager version
2
Options
4
Window component count
2
First window component
Resource ID (of 'hrct' or 'hdlg' resource)
2
Type of associated resource ('hrct' or 'hdlg')
4
Length of comparison string—or a windowKind value
2
Window title string for comparison
1 to 256
Alignment bytes
0 to 1
Last window component
Resource ID (of 'hrct' or 'hdlg' resource)
2
Type of associated resource ('hrct' or 'hdlg')
4
Length of comparison string—or a windowKind value
2
Window title string for comparison
1 to 256
Alignment bytes
0 to 1

If you examine a compiled version of an `'hwin'` resource, you find that the header component consists of the following elements:

- Help Manager version. The version of the Help Manager to use. This is usually specified in a Rez input file with the `HelpMgrVersion` constant.
- Options. The sum of the values of available options, described in “Specifying Options in Help Resources” beginning on page 3-25.
- Window component count. The number of window components defined in the rest of this resource. The Help Manager determines the end of the `'hwin'` resource by using this component count information.

If you examine a compiled version of an `'hwin'` resource, you find that a window component consists of the following elements:

- Resource ID. The ID of the associated resource (either `'hrct'` or `'hdlg'`) that specifies the help messages for the window.
- Type of associated resource. A resource type; either `'hrct'` or `'hdlg'`.
- Length of comparison string—or a `windowKind` value. If the integer in this element is positive, this is the number of characters used for matching this component to a window’s title. If the integer in this element is negative, this is a value used for matching this component to a window by the `windowKind` value in the window’s window record.
- Window title string. If the previous element is a positive integer, this element consists of characters that the Help Manager uses to match this component to a window by the window’s title. If the previous element is a negative integer, this is an empty string.
- Alignment bytes. Zero or one bytes used to make the window title string end on a word boundary.

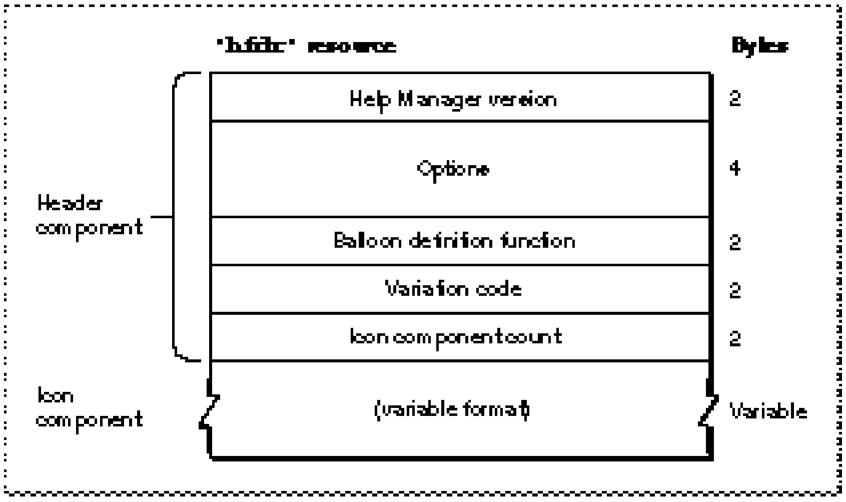
The Finder Icon Help Resource

The Help Manager displays default help messages for all Finder icon types. By creating a Finder icon help override resource, you can provide your own help message for the Help Manager to display when the user moves the cursor over your non-document icons. A Finder icon help resource is a resource of type `'hfdr'`. An `'hfdr'` resource must have a resource ID of –5696.

The format of a Rez input file for an `'hfdr'` resource differs from its compiled output form. This section describes the structure of a Rez-compiled `'hfdr'` resource. If you are concerned only with creating `'hfdr'` resources, see “Overriding Help Balloons for Non-Document Icons” on page 3-84 for a detailed description of how to use Rez input files to create an `'hfdr'` resource.

An 'hfdr' resource consists of a header component and one icon component. Figure 3-41 shows the general structure of a compiled 'hfdr' resource.

Figure 3-41 Structure of a compiled Finder icon help ('hfdr') resource



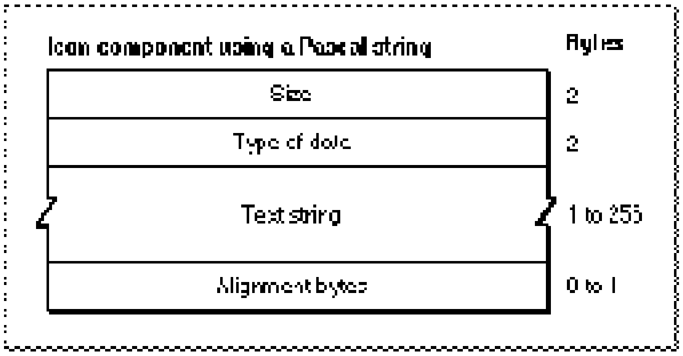
If you examine a compiled version of an 'hfdr' resource, you find that the header component consists of the following elements:

- Help Manager version. The version of the Help Manager to use. This is usually specified in a Rez input file with the `HelpMgrVersion` constant.
- Options. The sum of the values of available options, described in "Specifying Options in Help Resources" beginning on page 3-25.
- Balloon definition function. The resource ID of the window definition function used for drawing the help balloon. The standard balloon definition function is of type 'WDEF' with resource ID 126; this can be specified by the number 0 in the Rez input file.
- Variation code. A number signifying the preferred position of the help balloon relative to the hot rectangle. The balloon definition function draws the frame of the help balloon based on the variation code specified here. The eight variation codes and how they affect the standard balloon definition function are illustrated in Figure 3-4 on page 3-10.
- Icon component count. The value 1, because only one icon component can be defined in this resource.

The structure of the icon component depends on the identifier specified for that component. The identifiers used in a Rez input file are described in "Specifying the Format for Help Messages" on page 3-23.

Figure 3-42 shows the structure of an icon component that stores its help message as a Pascal string within the `'hfdr'` resource itself.

Figure 3-42 Structure of an `'hfdr'` component compiled with the `HMStringItem` identifier

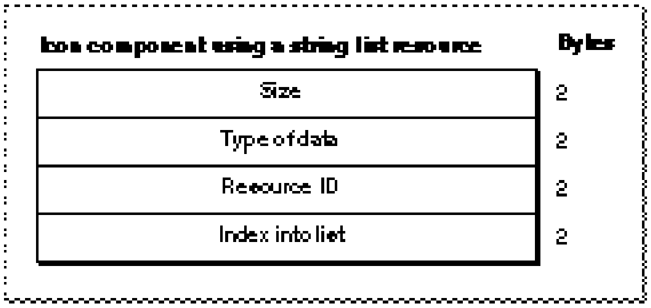


If you examine a compiled version of an `'hfdr'` resource, you find that a component identified in a Rez input file by the `HMStringItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data. The value 1 is specified here when the help messages are stored as a Pascal string within this component.
- Text string. The help message that the Help Manager displays when the user moves the cursor over the icon.
- Alignment bytes. Zero or one bytes used to make the previous text strings end on a word boundary.

Figure 3-43 shows the structure of an icon component that specifies its help message as a text string stored in a string list (`'STR#'`) resource.

Figure 3-43 Structure of an `'hfdr'` component compiled with the `HMStringResItem` identifier

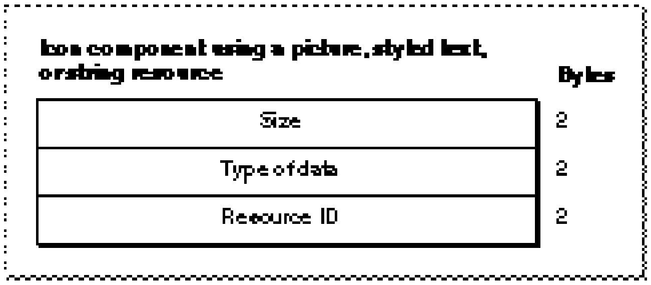


If you examine a compiled version of an `'hfdr'` resource, you find that a component identified in a Rez input file by the `HMStringResItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data. The value 3 is specified here when the help messages for this component are stored in string list (`'STR#'`) resources.
- Resource ID. The resource ID of an `'STR#'` resource.
- Index into the string list resource. A number used as an index to a particular text string within the `'STR#'` resource. The Help Manager displays this text string for the help message.

Figure 3-44 shows the structure of an icon component that specifies its help message in a picture (`'PICT'`) resource, in styled text (`'TEXT'` and `'styl'`) resources, or in a string (`'STR '`) resource.

Figure 3-44 Structure of an `'hfdr'` component compiled with the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier


Icon component using a picture, styled text, or string resource
Bytes
Size
2
Type of data
2
Resource ID
2

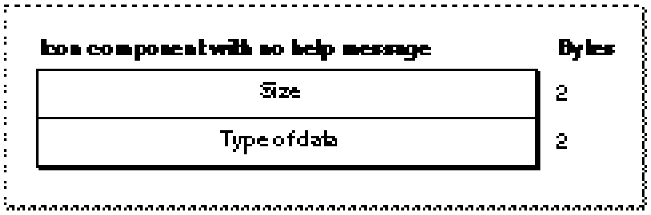
If you examine a compiled version of an `'hfdr'` resource, you find that a component identified in a Rez input file by either the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data.
  - The value 2 is specified here when the help message for this component is stored in a `'PICT'` resource.
  - The value 6 is specified here when the help message for this component is stored as styled text—that is, in both `'TEXT'` and `'styl'` resources.
  - The value 7 is specified here when the help message for this component is stored in an `'STR '` resource.

- Resource ID.
  - The resource ID of a `'PICT'` resource when the value 2 is specified as the type of data. The Help Manager displays the picture stored in this resource for the help message.
  - The resource ID common to both a `'TEXT'` and an `'styl'` resource when the value 6 is specified as the type of data. The Help Manager displays the styled text specified in these resources for the help message.
  - The resource ID of an `'STR '` resource when the value 7 is specified as the type of data. The Help Manager uses the text string stored in this resource for the help message.

Figure 3-45 shows the structure of an icon component that doesn't specify a help message.

Figure 3-45 Structure of an `'hfdr'` component compiled with the `HMSkipItem` identifier



If you examine a compiled version of an `'hfdr'` resource, you find that a component identified by the `HMSkipItem` identifier consists of the following elements:

- Size. The value 4, for the number of bytes contained in this component.
- Type of data. The value 256.

The Default Help Override Resource

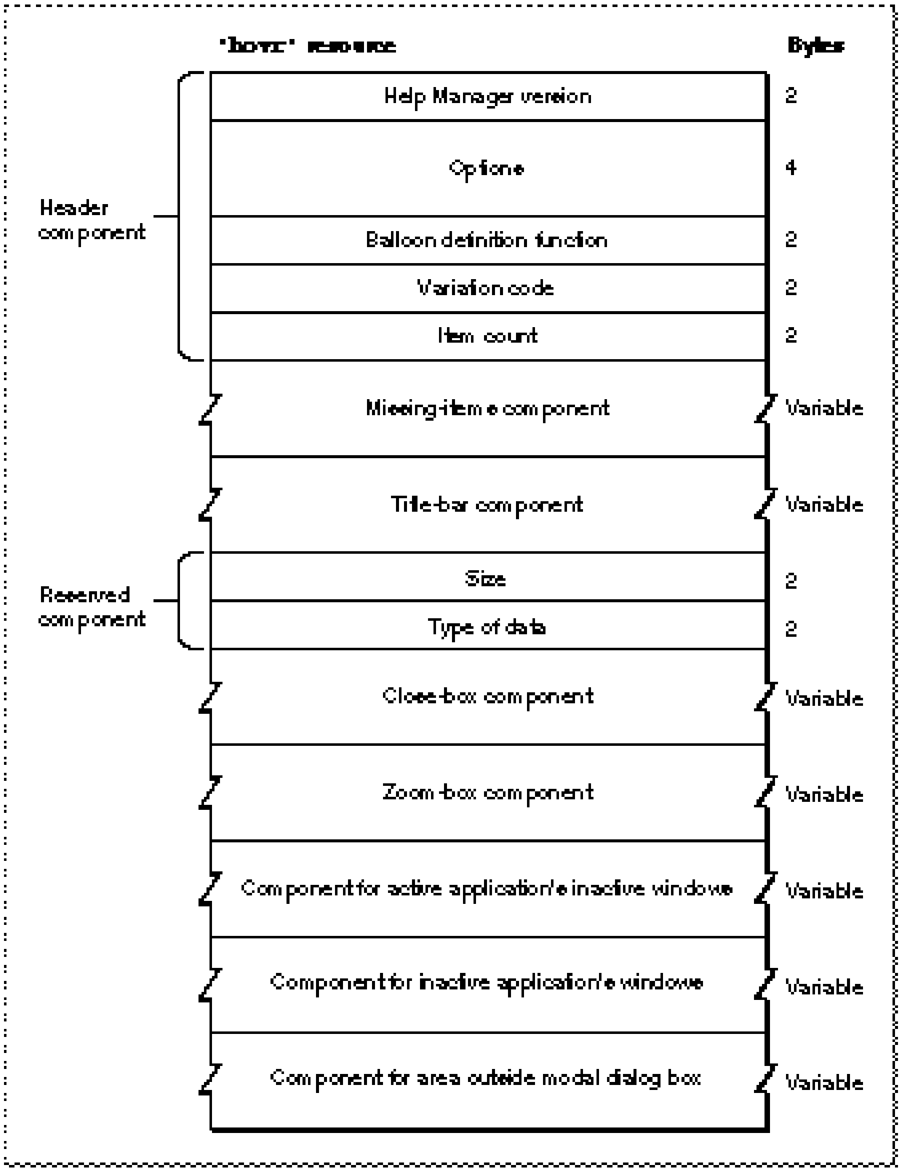
The Help Manager also provides default help balloons for the title bar and the close and zoom boxes of an active window, for the windows of inactive applications, for inactive windows of an active application, and for the area outside a modal dialog box.

Apple has researched and tested these help messages to ensure that they are as effective as possible for users. Normally, you don't need to override them. However, by creating a default help override resource you can override one or more of these defaults if absolutely necessary. A default help override resource is a resource of type `'hovr'`. The `'hovr'` resource must have a resource ID greater than 128.

The format of a Rez input file for an `'hovr'` resource differs from its compiled output form. This section describes the structure of a Rez-compiled `'hovr'` resource. If you are concerned only with creating `'hovr'` resources, see "Overriding Other Default Help Balloons" on page 3-87 for a detailed description of how to use Rez input files to create `'hovr'` resources.

An 'hovr' resource consists of a header component, a missing-items component, and seven additional components for various interface elements. Figure 3-46 shows the general structure of a compiled 'hovr' resource.

Figure 3-46 Structure of a compiled default help override ('hovr') resource



If you examine a compiled version of an `'hovr'` resource, you find that the header component consists of the following elements:

- Help Manager version. The version of the Help Manager to use. This is usually specified in a Rez input file with the `HelpMgrVersion` constant.
- Options. The sum of the values of available options, described in “Specifying Options in Help Resources” beginning on page 3-25.
- Balloon definition function. The resource ID of the window definition function used for drawing the help balloon. The standard balloon definition function is of type `'WDEF'` with resource ID 126; this can be specified by 0 in the Rez input file.
- Variation code. A number signifying the preferred position of the help balloon relative to the hot rectangle. The balloon definition function draws the frame of the help balloon based on the variation code specified here. The eight variation codes and how they affect the standard balloon definition function are illustrated in Figure 3-4 on page 3-10.
- Item count. The value 8 for the number of components defined in the rest of this resource.

The Help Manager uses the order of the components in this resource to determine their purposes.

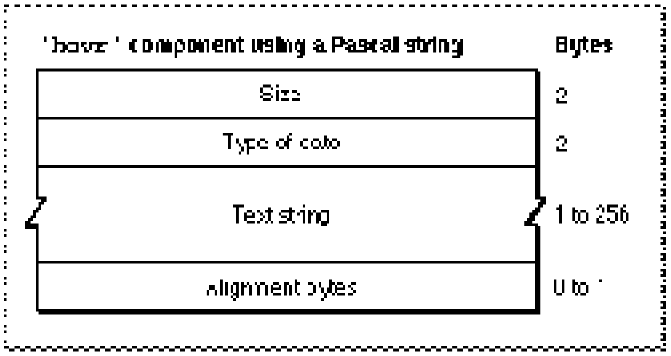
The structures of the remaining components depend on identifiers specified inside the components. The identifiers used in a Rez input file are described in “Specifying the Format for Help Messages” on page 3-23.

Each component can specify one help message, as listed here.

- Missing-items component. The Help Manager expects seven more components to follow, in the order listed here. If fewer than seven components are specified in the Rez input file, the Help Manager adds components to the end of the list until there are seven. Each component that the Help Manager adds uses the message specified in the missing-items component. The Help Manager also uses the missing-items component’s help message if the input file specifies an empty string or a resource ID of 0 for any other component’s help message.
- Title-bar component. The help message for title bar of the active window.
- Reserved component. This element is reserved and should have no help message. The `HMSkipItem` identifier should always be specified in the Rez input file for this component.
- Close-box component. The help message for the close box of the active window.
- Zoom-box component. The help message for the zoom box of the active window.
- Component for active application’s inactive windows. The help message for the inactive windows of the active application.
- Component for inactive applications’ windows. The help message for the windows of inactive applications.
- Component for area outside modal box. The help message for the desktop area outside a modal dialog box or an alert box.

Figure 3-47 shows the structure of an `'hovr'` component that stores its help message as a Pascal string within the `'hovr'` resource itself.

Figure 3-47 Structure of an `'hovr'` component compiled with the `HMStringItem` identifier

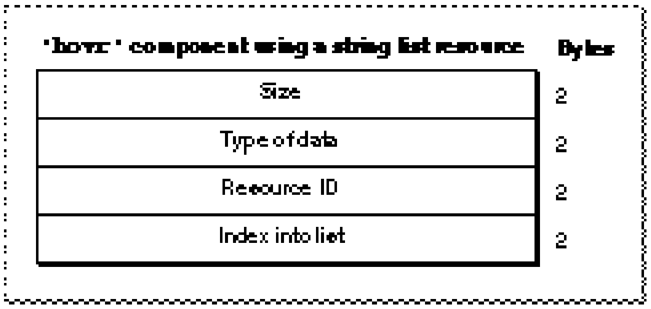


If you examine a compiled version of an `'hovr'` resource, you find that a component identified in a Rez input file by the `HMStringItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data. The value 1 is specified here when the help message is stored as a Pascal string within this component.
- Text string. The help message appropriate for the component (as previously described).
- Alignment bytes. Zero or one bytes used to make the text string end on a word boundary.

Figure 3-48 shows the structure of an `'hovr'` component that specifies its help message as a text string stored in a string list (`'STR#'`) resource.

Figure 3-48 Structure of an `'hovr'` component compiled with the `HMStringResItem` identifier

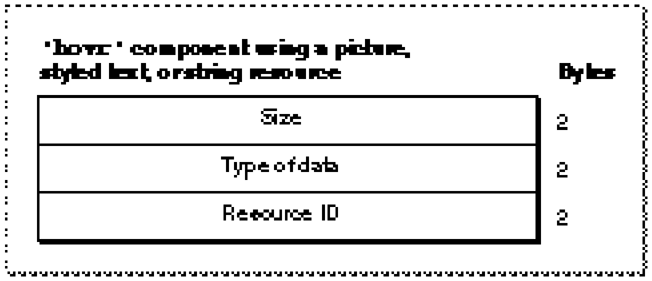


If you examine a compiled version of an `'hovr'` resource, you find that a component identified in a Rez input file by the `HMStringResItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data. The value 3 is specified here when the help message for this component is stored in a string list (`'STR#'`) resource.
- Resource ID. The resource ID of an `'STR#'` resource.
- Index into the string list resource. A number used as an index to a particular text string within the `'STR#'` resource. The Help Manager uses this text string for the help message of the appropriate component (as previously described).

Figure 3-49 shows the structure of an `'hovr'` component that specifies its help message in a picture (`'PICT'`) resource, in styled text (`'TEXT'` and `'styl'`) resources, or in a string (`'STR '`) resource.

Figure 3-49 Structure of an `'hovr'` component compiled with the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier


'hovr' component using a picture, styled text, or string resource
Bytes
Size
2
Type of data
2
Resource ID
2

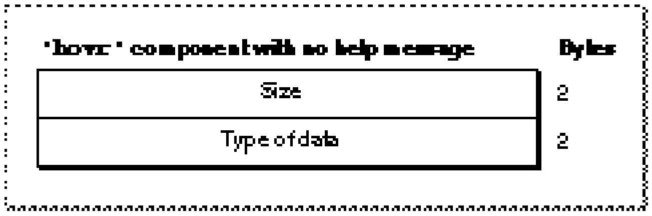
If you examine a compiled version of an `'hovr'` resource, you find that a component identified in a Rez input file by either the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier consists of the following elements:

- Size. The number of bytes contained in this component.
- Type of data.
  - The value 2 is specified here when the help message for this component is stored in a `'PICT'` resource.
  - The value 6 is specified here when the help message for this component is stored as styled text—that is, in both `'TEXT'` and `'styl'` resources.
  - The value 7 is specified here when the help message for this component is stored in an `'STR '` resource.

- Resource ID.
  - The resource ID of a `'PICT'` resource when the value 2 is specified as the type of data. The Help Manager displays the picture stored in this resource for the help message.
  - The resource ID common to both a `'TEXT'` and an `'styl'` resource when the value 6 is specified as the type of data. The Help Manager displays the styled text specified in these resources for the help message.
  - The resource ID of an `'STR '` resource when the value 7 is specified as the type of data. The Help Manager uses the text string stored in this resource for the help message.

Figure 3-50 shows the structure of an `'hovr'` component that doesn't specify a help message.

Figure 3-50 Structure of an `'hovr'` component compiled with the `HMSkipItem` identifier



If you examine a compiled version of an `'hovr'` resource, you find that a component identified in the Rez input file by the `HMSkipItem` identifier consists of the following elements:

- Size. The value 4, for the number of bytes contained in this component.
- Type of data. The value 256.

Summary of the Help Manager

Pascal Summary

Constants

```
CONST
   gestaltHelpMgrAttr      = 'help';   {Gestalt selector}
   gestaltHelpMgrPresent   =  0;       {if this bit is set, then }
                                       { Help Manager is present}
   hmBalloonHelpVersion    = $0002;    {Help Manager version}
   kBalloonWDEFID          = 126;      {resource ID of standard balloon }
                                       { 'WDEF' function}
   kHMHelpID               = -5696;    {ID of various Help Manager }
                                       { resources (in Pack14 range); }
                                       { also used for 'hfdr' resource ID}

   {Help menu constants}
   kHMAboutHelpItem        = 1;        {About Balloon Help menu item}
   kHMHelpMenuID           = -16490;   {Help menu resource ID}
   kHMShowBalloonsItem     = 3;        {Show/Hide Balloons menu item}

   {HelpItem type for 'DITL' resources}
   helpItem                = 1;        {help item}

   {option bits for help resources}
   hmDefaultOptions        = 0;        {use defaults}
   hmUseSubID              = 1;        {use subrange resource IDs }
                                       { for owned resources}
   hmAbsoluteCoords        = 2;        {ignore coords of window }
                                       { origin and treat upper-left }
                                       { corner of window as 0,0}
   hmSaveBitsNoWindow      = 4;        {don't create window; save }
                                       { bits; no update event}
   hmSaveBitsWindow        = 8;        {save bits behind window and }
                                       { generate update event}
   hmMatchInTitle          = 16;       {match window by string }
                                       { anywhere in title string}
```

```
{constants for hmmHelpType field of HMMessageRecord}
khmmString              = 1;        {Pascal string}
khmmPict                = 2;        {'PICT' resource ID}
khmmStringRes           = 3;        {'STR#' res ID and index}
khmmTEHandle            = 4;        {TextEdit handle}
khmmPictHandle          = 5;        {picture handle}
khmmTERes               = 6;        {'TEXT' and 'styl' resource ID}
khmmSTRRes              = 7;        {'STR ' resource ID}
{resource types for styled text in resources}
kHMTETextResType        = 'TEXT';   {'TEXT' resource type}
kHMTEStyleResType       = 'styl';   {'styl' resource type}

{constants for whichState parameter when extracting help }
{ message records from 'hmnu' and 'hdlg' resources}
kHMEnabledItem          = 0;        {enabled state for menu items; }
                                    { contrlHilite value of 0 for }
                                    { controls}
kHMDisabledItem         = 1;        {disabled state for menu items; }
                                    { contrlHilite value of 255 for }
                                    { controls}
kHMCheckedItem          = 2;        {enabled-and-checked state for }
                                    { menu items; contrlHilite }
                                    { value of 1 for controls that }
                                    { are "on"}
kHMOtherItem            = 3;        {enabled-and-marked state for }
                                    { menu items; contrlHilite }
                                    { value between 2 and 253 for }
                                    { controls}

{resource types for whichType parameter used when extracting }
{ help message}
kHMMenuResType          = 'hmnu';   {menu help resource type}
kHMDialogResType        = 'hdlg';   {dialog help resource type}
kHMWindListResType      = 'hwin';   {window help resource type}
kHMRectListResType      = 'hrct';   {rectangle help resource type}
kHMOverrideResType      = 'hovr';   {help override resource type}
kHMFinderApplResType    = 'hfdr';   {app icon help resource type}

{constants for method parameter in HMShowBalloon}
kHMRegularWindow        = 0;        {don't save bits; just update}
kHMSaveBitsNoWindow     = 1;        {save bits; don't do update}
kHMSaveBitsWindow       = 2;        {save bits; do update event}
```

```
   {constants for help types in 'hmnu', 'hdlg', 'hrct', 'hovr', and }
   { 'hfdr' resources--useful only for walking these resources}
   kHMStringItem           = 1;        {Pascal string}
   kHMPictItem             = 2;        {'PICT' resource ID}
   kHMStringResItem        = 3;        {'STR#' resource ID & index}
   kHMTEResItem            = 6;        {'TEXT' & 'styl' resource ID}
   kHMSTRResItem           = 7;        {'STR ' resource ID}
   kHMSkipItem             = 256;      {don't display a balloon}
   kHMCompareItem          = 512;      {for 'hmnu', use help message }
                                       { if menu item matches string}
   kHMNamedResourceItem    = 1024;     {for 'hmnu', use menu item to }
                                       { get a named resource}
   kHMTrackCntlItem        = 2048;     {reserved}
```

Data Types

```
TYPE  HMStringResType   =                 {Help Manager string list record}
      RECORD
         hmmResID:      Integer;          {'STR#' resource ID}
         hmmIndex:      Integer;          {index of string}
      END;

      HMMessageRecPtr   = ^HMMessageRecord;
      HMMessageRecord   =                             {help message record}
      RECORD
         hmmHelpType:         Integer;                {type of next field}
         CASE Integer OF
            khmmString:       (hmmString: Str255);    {Pascal string}
            khmmPict:         (hmmPict: Integer);     {'PICT' resource ID}
            khmmStringRes:    (hmmStringRes: HMStringResType);
                                                      {'STR#' resource }
                                                      { ID and index}
            khmmTEHandle:     (hmmTEHandle: TEHandle);{TextEdit handle}
            khmmPictHandle:   (hmmPictHandle: PicHandle);
                                                      {picture handle}
            khmmTERes:        (hmmTERes: Integer);    {'TEXT'/'styl' }
                                                      { resource ID}
            khmmSTRRes:       (hmmSTRRes: Integer)    {'STR ' resource ID}
      END;
```

Help Manager Routines

Determining Help Balloon Status

```
FUNCTION HMGetBalloons      : Boolean;
FUNCTION HMIsBalloon        : Boolean;
```

Displaying and Removing Help Balloons

```
FUNCTION HMShowBalloon      (aHelpMsg: HMMessageRecord; tip: Point;
                             alternateRect: RectPtr; tipProc: Ptr;
                             theProc: Integer; variant: Integer;
                             method: Integer): OSErr;
FUNCTION HMShowMenuBalloon  (itemNum: Integer; itemMenuID: Integer;
                             itemFlags: LongInt; itemReserved: LongInt;
                             tip: Point; alternateRect: RectPtr;
                             tipProc: Ptr; theProc: Integer;
                             variant: Integer): OSErr;
FUNCTION HMRemoveBalloon    : OSErr;
```

Enabling and Disabling Balloon Help Assistance

```
FUNCTION HMSetBalloons      (flag: Boolean): OSErr;
```

Adding Items to the Help Menu

```
FUNCTION HMGetHelpMenuHandle
                            (VAR mh: MenuHandle): OSErr;
```

Getting and Setting the Font Name and Size

```
FUNCTION HMGetFont          (VAR font: Integer): OSErr;
FUNCTION HMGetFontSize      (VAR fontSize: Integer): OSErr;
FUNCTION HMSetFont          (font: Integer): OSErr;
FUNCTION HMSetFontSize      (fontSize: Integer): OSErr;
```

Setting and Getting Information for Help Resources

```
FUNCTION HMSetMenuResID     (menuID: Integer; resID: Integer): OSErr;
FUNCTION HMGetMenuResID     (menuID: Integer; VAR resID: Integer): OSErr;
FUNCTION HMScanTemplateItems
                            (whichID: Integer; whichResFile: Integer;
                             whichType: ResType): OSErr;
FUNCTION HMSetDialogResID   (resID: Integer): OSErr;
FUNCTION HMGetDialogResID   (VAR resID: Integer): OSErr;
```

Determining the Size of a Help Balloon

```
FUNCTION HMBalloonRect      (aHelpMsg: HMMessageRecord;
                             VAR coolRect: Rect): OSErr;
FUNCTION HMBalloonPict      (aHelpMsg: HMMessageRecord;
                             VAR coolPict: PicHandle): OSErr;
FUNCTION HMGetBalloonWindow
                            (VAR window: WindowPtr): OSErr;
```

Getting the Message of a Help Balloon

```
FUNCTION HMExtractHelpMsg   (whichType: ResType;
                             whichResID: Integer; whichMsg: Integer;
                             whichState: Integer;
                             VAR aHelpMsg: HMMessageRecord): OSErr;
FUNCTION HMGetIndHelpMsg    (whichType: ResType;
                             whichResID: Integer; whichMsg: Integer;
                             whichState: Integer;
                             VAR options: LongInt; VAR tip: Point;
                             VAR altRect: Rect; VAR theProc: Integer;
                             VAR variant: Integer;
                             VAR aHelpMsg: HMMessageRecord;
                             VAR count: Integer): OSErr;
```

Application-Defined Routines

```
FUNCTION MyBalloonDef       (variant: Integer; theBalloon: WindowPtr;
                             message: Integer; param: LongInt): LongInt;
FUNCTION MyTip              (tip: Point; structure: RgnHandle;
                             VAR r: Rect; VAR variant: Integer): OSErr;
```

C Summary

Constants

```
enum {
   #define gestaltHelpMgrAttr  'help'  /*Gestalt selector*/
   gestaltHelpMgrPresent      =  0     /*if this bit is set, then */
                                       /* Help Manager is present*/
};
enum {
   hmBalloonHelpVersion    = 0x0002,   /*Help Manager version*/
   kBalloonWDEFID          = 126,      /*resource ID of standard balloon */
                                       /* 'WDEF' function*/
   kHMHelpID               = -5696,    /*ID of various Help Manager */
                                       /* resources (in Pack14 range); */
                                       /* also used for 'hfdr' resource ID*/

   /*Help menu constants*/
   kHMAboutHelpItem        = 1,        /*About Balloon Help menu item*/
   kHMHelpMenuID           = -16490,   /*Help menu resource ID*/
   kHMShowBalloonsItem     = 3,        /*Show/Hide Balloons menu item*/

   /*help item type for 'DITL' resources*/
   HelpItem                = 1,        /*help item*/

   /*option bits for help resources*/
   hmDefaultOptions        = 0,        /*use defaults*/
   hmUseSubID              = 1,        /*use subrange resource IDs */
                                       /* for owned resources*/
   hmAbsoluteCoords        = 2         /*ignore coords of window */
                                       /* origin and treat upper-left */
                                       /* corner of window as 0,0*/
};
enum {
   hmSaveBitsNoWindow      = 4,        /*don't create window; save */
                                       /* bits; no update event*/
   hmSaveBitsWindow        = 8,        /*save bits behind window and */
                                       /* generate update event*/
   hmMatchInTitle          = 16,       /*match window by string */
                                       /* anywhere in title string*/

   /*constants for hmmHelpType field of HMMessageRecord*/
   khmmString              = 1,        /*Pascal string*/
   khmmPict                = 2,        /*'PICT' resource ID*/
   khmmStringRes           = 3,        /*'STR#' res ID and index*/
   khmmTEHandle            = 4,        /*TextEdit handle*/
   khmmPictHandle          = 5,        /*picture handle*/
   khmmTERes               = 6,        /*'TEXT' and 'styl' resource ID*/
   khmmSTRRes              = 7,        /*'STR ' resource ID*/
   /*resource types for styled text in resources*/
   #define kHMTETextResType    'TEXT'  /*'TEXT' resource type*/
   #define kHMTEStyleResType   'styl'  /*'styl' resource type*/
   /*constants for whichState parameter when extracting help */
   /* message records from 'hmnu' and 'hdlg' resources*/
   kHMEnabledItem          = 0,        /*enabled state for menu items; */
                                       /* contrlHilite value of 0 for */
                                       /* controls*/
};
enum {
   kHMDisabledItem         = 1,        /*disabled state for menu items; */
                                       /* contrlHilite value of 255 for */
                                       /* controls*/
   kHMCheckedItem          = 2,        /*enabled-and-checked state for */
                                       /* menu items; contrlHilite */
                                       /* value of 1 for controls that */
                                       /* are "on"*/
   kHMOtherItem            = 3,        /*enabled-and-marked state for */
                                       /* menu items; contrlHilite */
                                       /* value between 2 and 253 for */
                                       /* controls*/

   /*resource types for whichType parameter used when extracting */
   /* help message*/
   #define kHMMenuResType        'hmnu'   /*menu help resource type*/
   #define kHMDialogResType      'hdlg'   /*dialog help resource type*/
   #define kHMWindListResType    'hwin'   /*window help resource type*/
   #define kHMRectListResType    'hrct'   /*rectangle help resource type*/
   #define kHMOverrideResType    'hovr'   /*help override resource type*/
   #define kHMFinderApplResType  'hfdr'   /*app icon help resource type*/

   /*constants for method parameter in HMShowBalloon*/
   kHMRegularWindow        = 0,        /*don't save bits; just update*/
   kHMSaveBitsNoWindow     = 1,        /*save bits; don't do update*/
   kHMSaveBitsWindow       = 2         /*save bits; do update event*/
};
enum {
   /*constants for help types in 'hmnu', 'hdlg', 'hrct', 'hovr', and */
   /* 'hfdr' resources--useful only for walking these resources*/
   kHMStringItem           = 1,        /*Pascal string*/
   kHMPictItem             = 2,        /*'PICT' resource ID*/
   kHMStringResItem        = 3,        /*'STR#' resource ID & index*/
   kHMTEResItem            = 6,        /*'TEXT' & 'styl' resource ID*/
   kHMSTRResItem           = 7,        /*'STR ' resource ID*/
   kHMSkipItem             = 256,      /*don't display a balloon*/
   kHMCompareItem          = 512,      /*for 'hmnu', use help message */
                                       /* if menu item matches string*/
   kHMNamedResourceItem    = 1024,     /*for 'hmnu', use menu item to */
                                       /* get a named resource*/
   kHMTrackCntlItem        = 2048      /*reserved*/
};
```

Data Types

```
struct HMStringResType {      /*Help Manager string list record*/
      short    hmmResID;      /*'STR#' resource ID*/
      short    hmmIndex;      /*index of string*/
};
typedef struct HMStringResType HMStringResType;

struct HMMessageRecord {      /*help message record*/
   short hmmHelpType;         /*type of next field*/
   union {
      char              hmmString[256];   /*Pascal string*/
      short             hmmPict;          /*'PICT' resource ID*/
      Handle            hmmTEHandle;      /*TextEdit handle*/
      HMStringResType   hmmStringRes;     /*'STR#' resource ID and index*/
      short             hmmPictRes;       /*unused*/
      Handle            hmmPictHandle;    /*picture handle*/
      short             hmmTERes;         /*'TEXT'/'styl' resource ID*/
      short             hmmSTRRes;        /*'STR ' resource ID*/
   } u;
};
typedef struct HMMessageRecord HMMessageRecord;
typedef HMMessageRecord *HMMessageRecPtr;
```

Help Manager Routines

Determining Help Balloon Status

```
pascal Boolean HMGetBalloons
                            (void);
pascal Boolean HMIsBalloon  (void);
```

Displaying and Removing Help Balloons

```
pascal OSErr HMShowBalloon  (const HMMessageRecord *aHelpMsg, Point tip,
                             RectPtr alternateRect, Ptr tipProc,
                             short theProc, short variant, short method);
pascal OSErr HMShowMenuBalloon
                            (short itemNum, short itemMenuID,
                             long itemFlags, long itemReserved,
                             Point tip, RectPtr alternateRect,
                             Ptr tipProc, short theProc, short variant);
pascal OSErr HMRemoveBalloon
                            (void);
```

Enabling and Disabling Balloon Help Assistance

```
pascal OSErr HMSetBalloons  (Boolean flag);
```

Adding Items to the Help Menu

```
pascal OSErr HMGetHelpMenuHandle
                            (MenuHandle *mh);
```

Getting and Setting the Font Name and Size

```
pascal OSErr HMGetFont      (short *font);
pascal OSErr HMGetFontSize  (short *fontSize);
pascal OSErr HMSetFont      (short font);
pascal OSErr HMSetFontSize  (short fontSize);
```

Setting and Getting Information for Help Resources

```
pascal OSErr HMSetMenuResID
                            (short menuID, short resID);
pascal OSErr HMGetMenuResID
                            (short menuID, short *resID);
pascal OSErr HMScanTemplateItems
                            (short whichID, short whichResFile,
                             ResType whichType);
pascal OSErr HMSetDialogResID
                            (short resID);
pascal OSErr HMGetDialogResID
                            (short *resID);
```

Determining the Size of a Help Balloon

```
pascal OSErr HMBalloonRect  (const HMMessageRecord *aHelpMsg,
                             Rect *coolRect);
pascal OSErr HMBalloonPict  (const HMMessageRecord *aHelpMsg,
                             PicHandle *coolPict);
pascal OSErr HMGetBalloonWindow
                            (WindowPtr *window);
```

Getting the Message of a Help Balloon

```
pascal OSErr HMExtractHelpMsg
                            (ResType whichType, short whichResID,
                             short whichMsg, short whichState,
                             HMMessageRecord *aHelpMsg);
pascal OSErr HMGetIndHelpMsg
                            (ResType whichType, short whichResID,
                             short whichMsg, short whichState,
                             long *options, Point *tip, Rect *altRect,
                             short *theProc, short *variant,
                             HMMessageRecord *aHelpMsg, short *count);
```

Application-Defined Routines

```
pascal long MyBalloonDef    (short variant, WindowPtr theBalloon,
                             short message, long param);
pascal OSErr MyTip          (Point tip, RgnHandle structure,
                             Rect *r, short *variant);
```

Assembly-Language Summary

Data Structures

Help Message Data Structure

0	`hmmHelpType`	word	Resource type
2	`hmmHelpMessage`	variable	Help balloon message

Trap Macros

Trap Macros Requiring Routine Selectors

`_Pack14`

Selector	Routine
$0002	`HMRemoveBalloon`
$0003	`HMGetBalloons`
$0007	`HMIsBalloon`
$0104	`HMSetBalloons`
$0108	`HMSetFont`
$0109	`HMSetFontSize`
$010C	`HMSetDialogResID`
$0200	`HMGetHelpMenuHandle`
$020A	`HMGetFont`
$020B	`HMGetFontSize`
$020D	`HMSetMenuResID`
$0213	`HMGetDialogResID`
$0215	`HMGetBalloonWindow`
$0314	`HMGetMenuResID`
$040E	`HMBalloonRect`
$040F	`HMBalloonPict`
$0410	`HMScanTemplateItems`
$0711	`HMExtractHelpMsg`
$0B01	`HMShowBalloon`
$0E05	`HMShowMenuBalloon`
$1306	`HMGetIndHelpMsg`

Result Codes

`noErr`	0	No error
`fnOpnErr`	–38	File not open
`paramErr`	–50	Error in parameter list
`memFullErr`	–108	Not enough room in heap zone
`resNotFound`	–192	Unable to read resource
`hmHelpDisabled`	–850	Help balloons are not enabled
`hmBalloonAborted`	–853	Because of constant cursor movement, the help balloon wasn’t displayed
`hmSameAsLastBalloon`	–854	Menu and item are same as previous menu and item
`hmHelpManagerNotInited`	–855	Help menu not set up
`hmSkippedBalloon`	–857	No help message to fill in
`hmWrongVersion`	–858	Wrong version of Help Manager resource
`hmUnknownHelpType`	–859	Help message record contained a bad type
`hmOperationUnsupported`	–861	Invalid value passed in the method parameter
`hmNoBalloonUp`	–862	No balloon showing
`hmCloseViewActive`	–863	Balloon can’t be removed because Close View is in use