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.

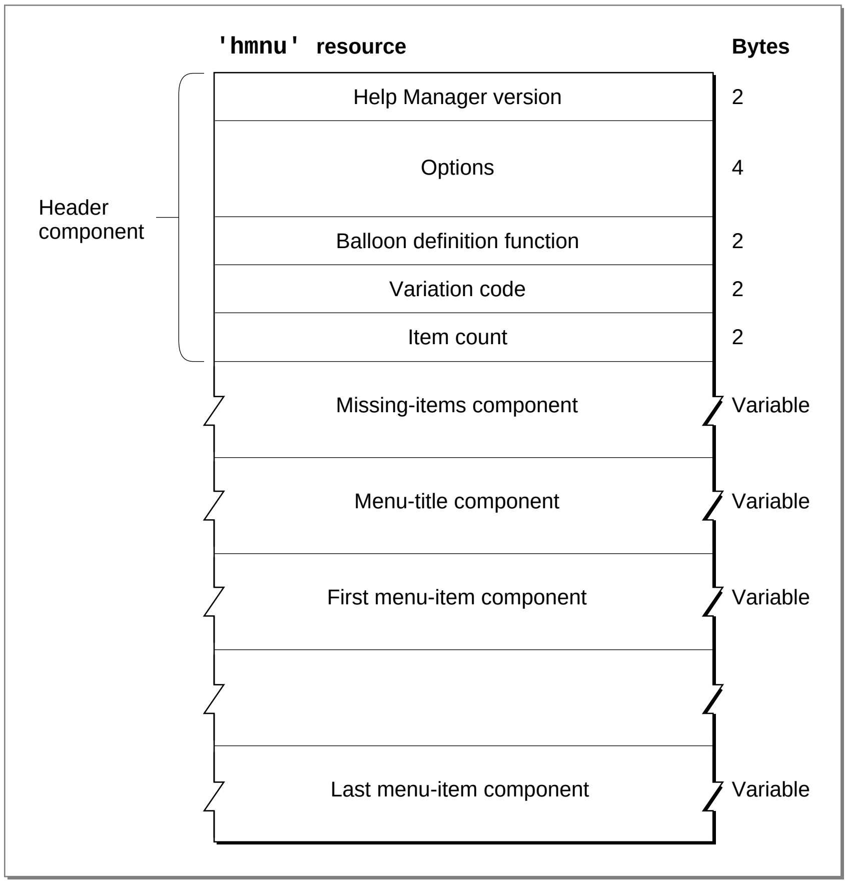
The Menu Help Resource

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.

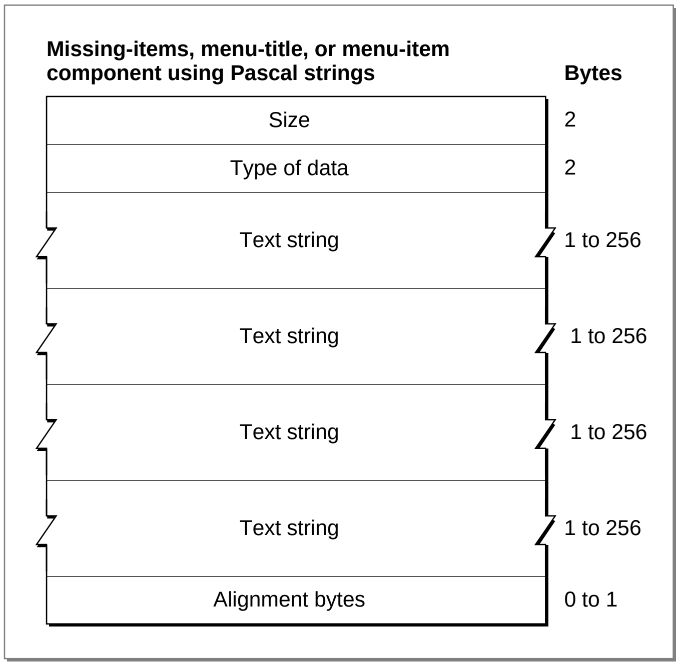
- Fourth help message.
  - In the missing-items component, this is the help message for missing enabled-and-marked items.
  - In the menu-title component, this is the help message for all menu items when system software dims them at the appearance of an alert box or a modal dialog box.
  - In all subsequent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

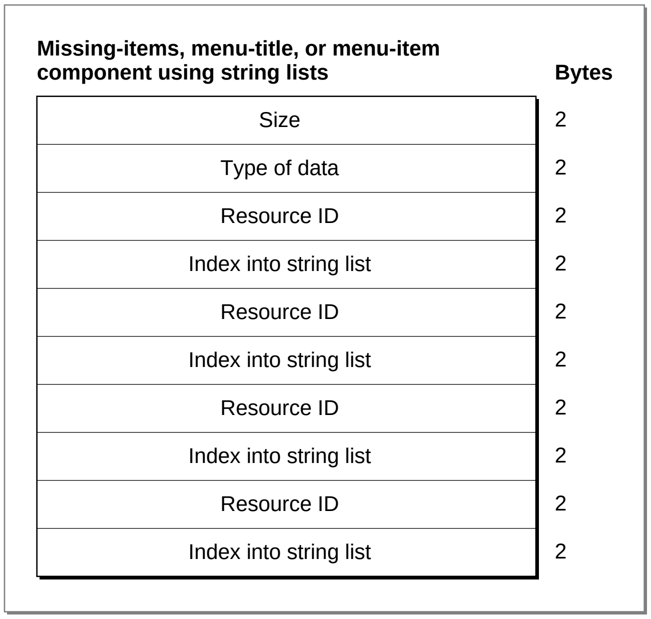


If you examine a compiled version of an `'hmnu'` resource, you find that a component identified in a Rez input file by the `HMStringItem` identifier consists of the following elements:

- 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



Missing-items, menu-title, or menu-item component using string lists	**Bytes**
Size	2
Type of data	2
Resource ID	2
Index into string list	2
Resource ID	2
Index into string list	2
Resource ID	2
Index into string list	2
Resource ID	2
Index into string list	2

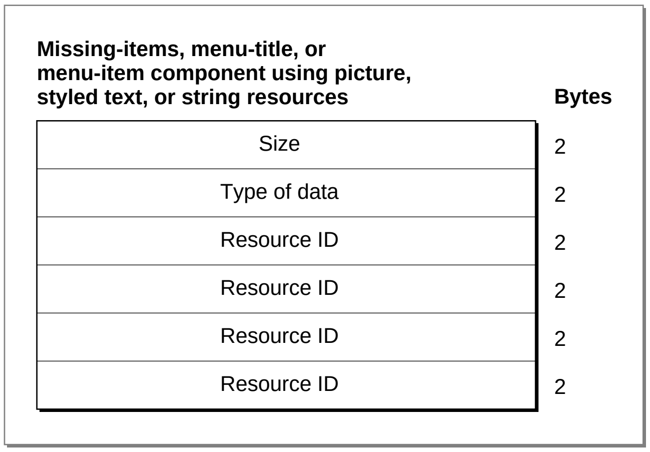
If you examine a compiled version of an `'hmnu'` resource, you find that a component identified in a Rez input file by the `HMStringResItem` identifier consists of the following elements:

- 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

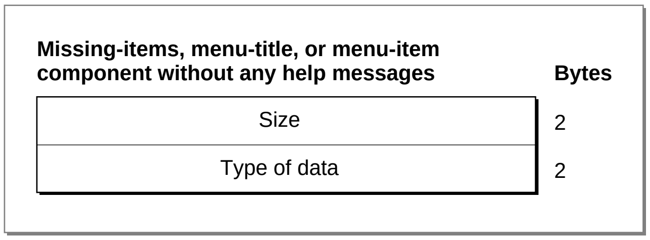
If you examine a compiled version of an `'hmnu'` resource, you find that a component identified in a Rez input file by either the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier consists of the following elements:

- 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

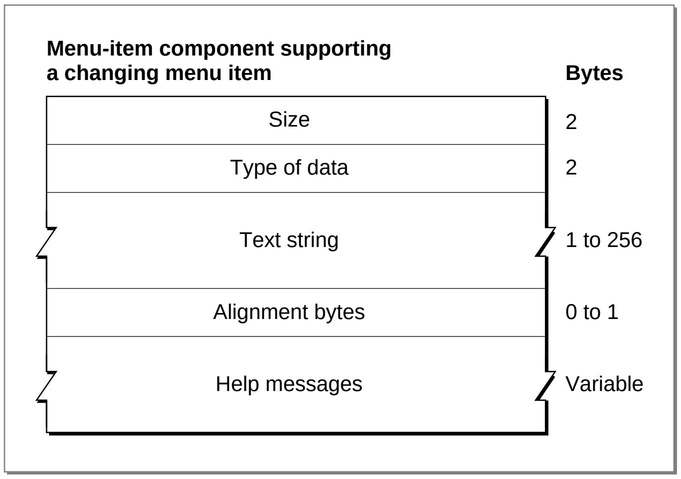

Missing-items, menu-title, or menu-item component without any help messages
Bytes
Size
2
Type of data
2

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



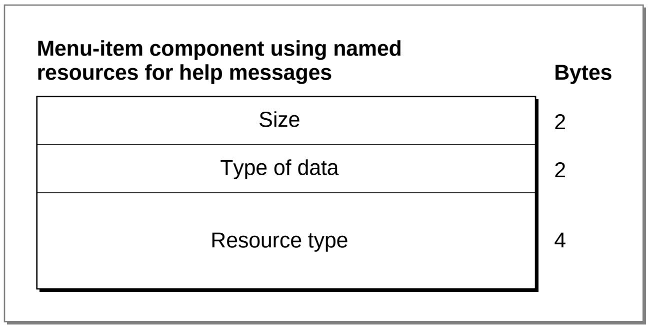
If you examine a compiled version of an `'hmnu'` resource, you find that a component identified in a Rez input file by the `HMCompareItem` identifier consists of these elements:

- 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.

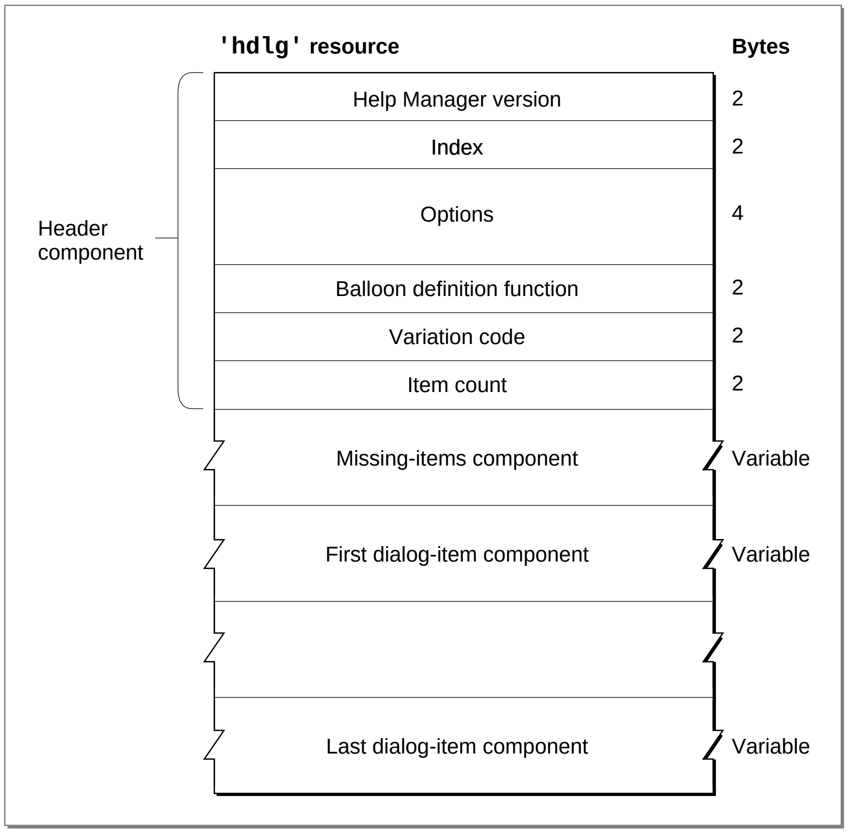
To associate an `'hdlg'` resource with a particular alert box or dialog box, either you must include an item of type `HelpItem` in the box's item list (`'DITL'`) resource, or you must create an `'hwin'` resource. Listing 3-8 on page 3-59 shows how to use an item of type `HelpItem`—and Listing 3-10 on 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:

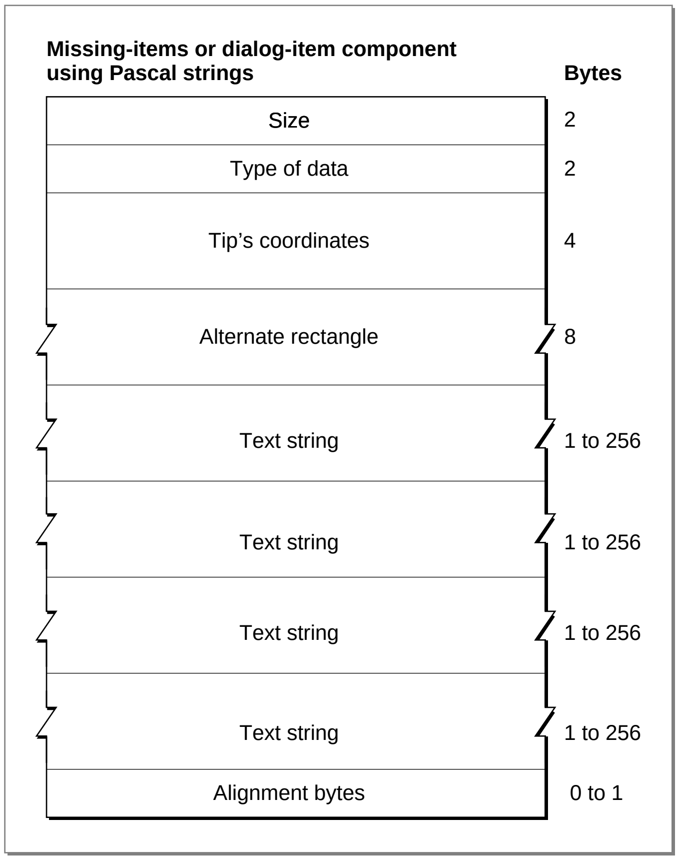
- First help message.
  - In the missing-items component, this is the help message both for missing, active, unselected controls (that is, those with highlight values of 0) and for missing enabled items that are not controls.
  - In dialog-item components, this is the help message for an active, unselected 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



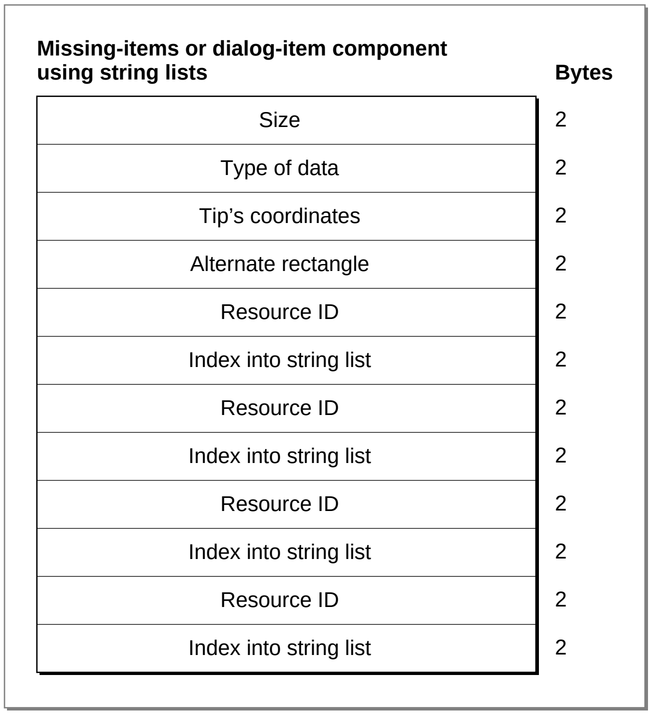
If you examine a compiled version of an `'hdlg'` resource, you find that a component identified in a Rez input file by the `HMStringItem` identifier consists of the following elements:

- 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



Missing-items or dialog-item component using string lists	Bytes
Size	2
Type of data	2
Tip's coordinates	2
Alternate rectangle	2
Resource ID	2
Index into string list	2
Resource ID	2
Index into string list	2
Resource ID	2
Index into string list	2
Resource ID	2
Index into string list	2

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:

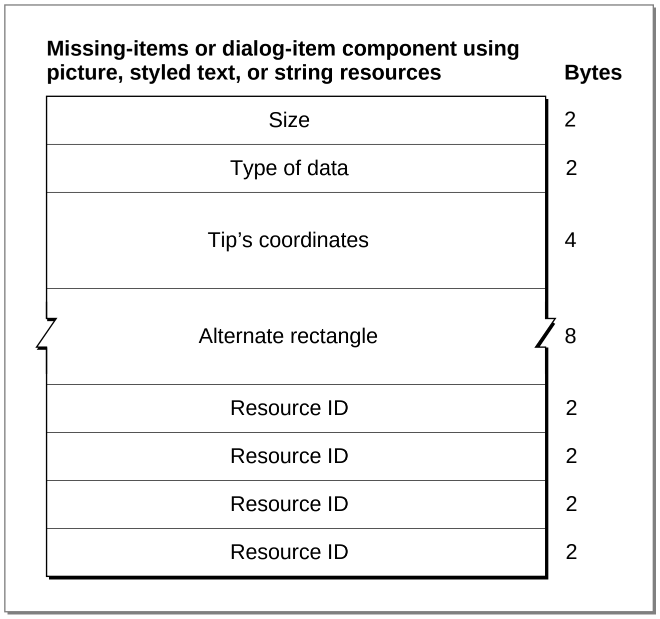
- Size. The number of bytes contained in this component.
- Type of data. The value 3 is specified here when the help messages for this component are stored in string list (`'STR#'`) resources.
- Tip's coordinates. The coordinates of the help balloon's tip. The tip's coordinates are local to the item's 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



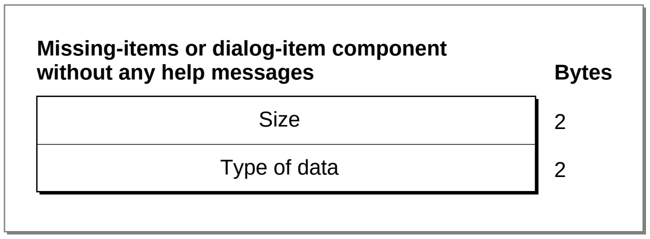
If you examine a compiled version of an `'hdlg'` resource, you find that a component identified in a Rez input file by either the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier consists of the following elements:

- 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

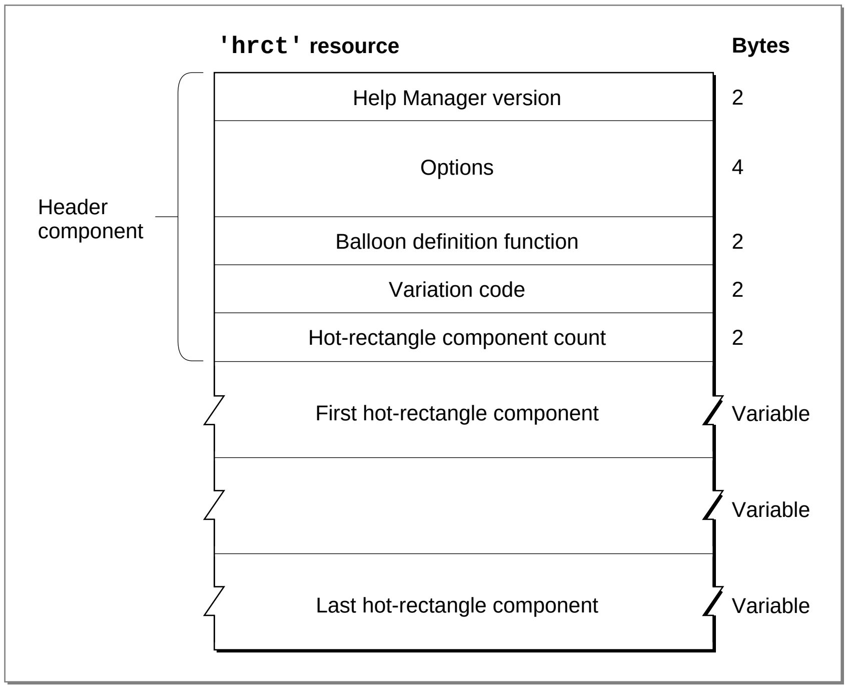
You can use a rectangle help resource to define hot rectangles for displaying help balloons within a static window, and to specify the help messages for those balloons. A rectangle help resource is a resource of type `'hrct'`. All `'hrct'` resources must have resource IDs greater than 128.

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:

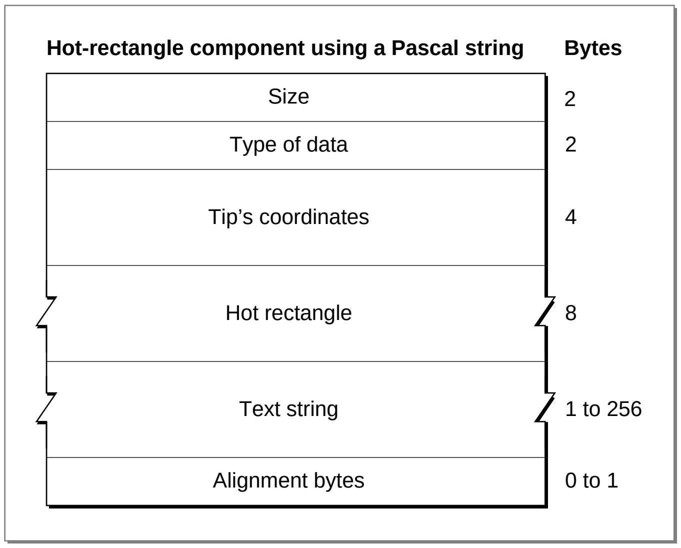
- Help Manager version. The version of the Help Manager to use. This is usually specified in a Rez input file with the `HelpMgrVersion` constant.
- Options. The sum of the values of available options, described in "Specifying Options in Help Resources" beginning on page 3-25.
- Balloon definition function. 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



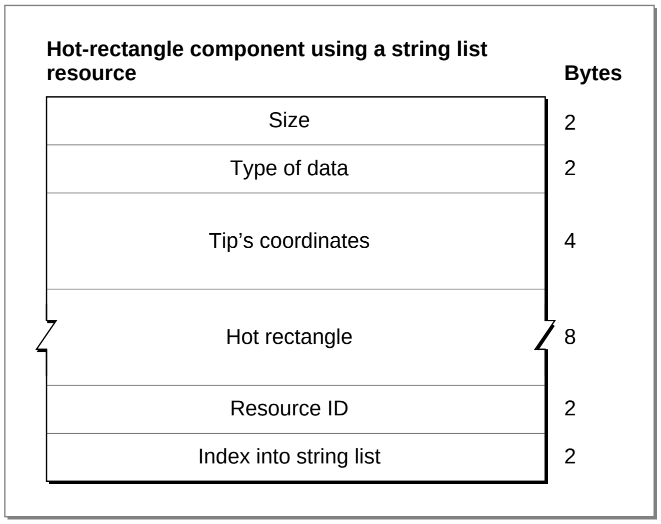
If you examine a compiled version of an `'hrct'` resource, you find that a component identified in a Rez input file by the `HMStringItem` identifier consists of the following elements:

- 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

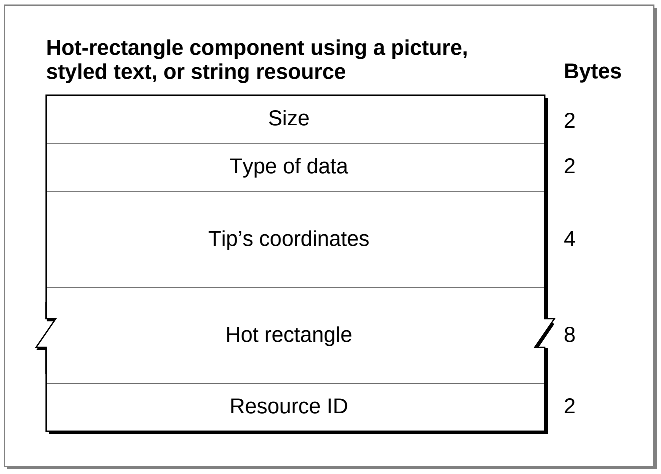


If you examine a compiled version of an `'hrct'` resource, you find that a component identified in a Rez input file by the `HMStringResItem` identifier consists of the following elements:

- 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

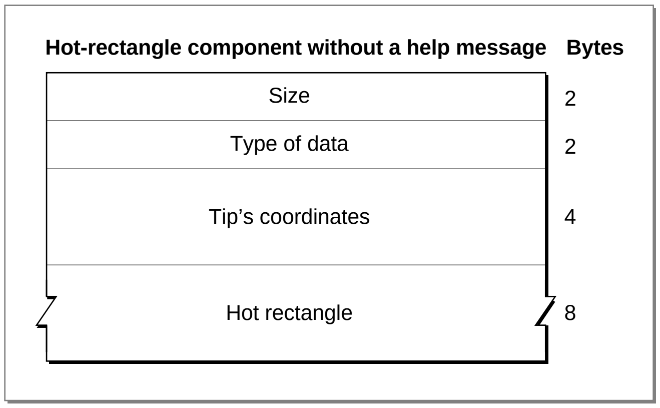
If you examine a compiled version of an `'hrct'` resource, you find that a component identified in a Rez input file by either the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier consists of the following elements:

- 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

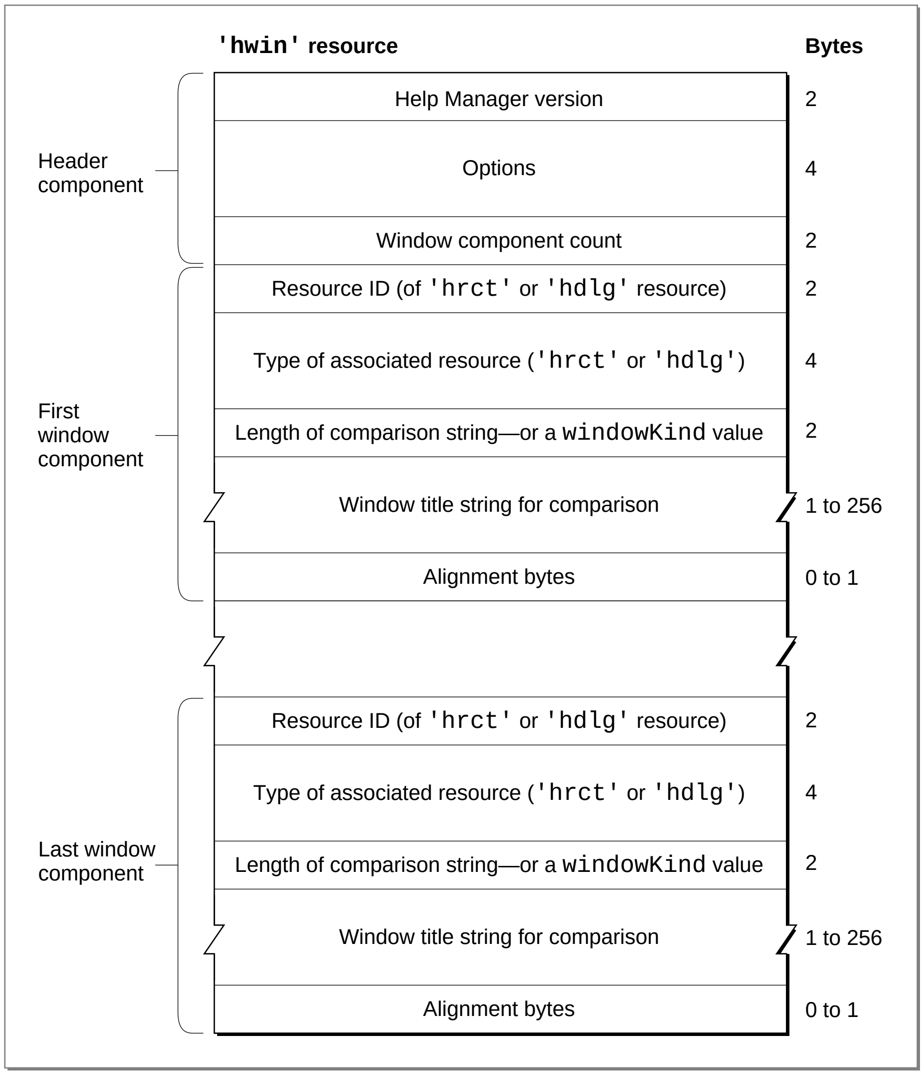
To associate the help balloons defined in an `'hrct'` resource with a particular window, you must create a window help resource. Unless you include an item of type `HelpItem` in an item list resource, you also must create a window help resource to associate an `'hdlg'` resource with a particular alert 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.

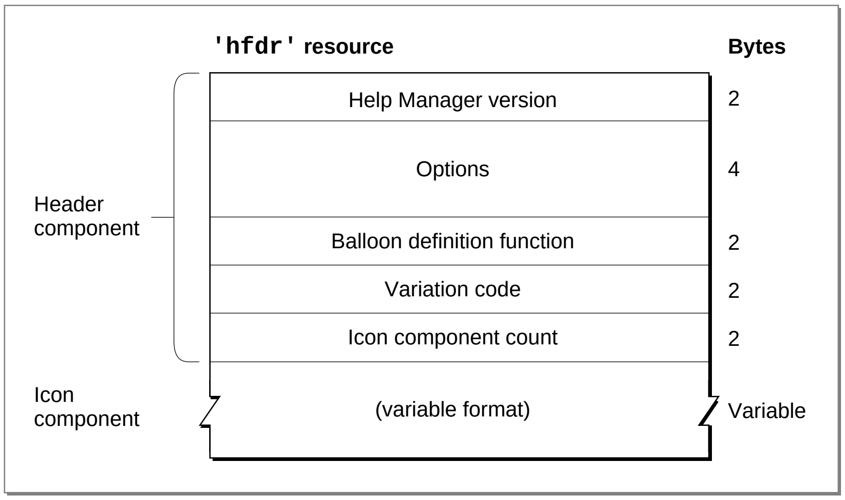
The Finder Icon Help Resource

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



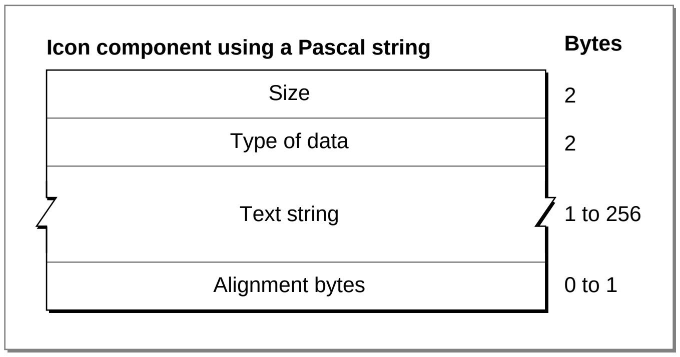
If you examine a compiled version of an `'hfdr'` resource, you find that the header component consists of the following elements:

- 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

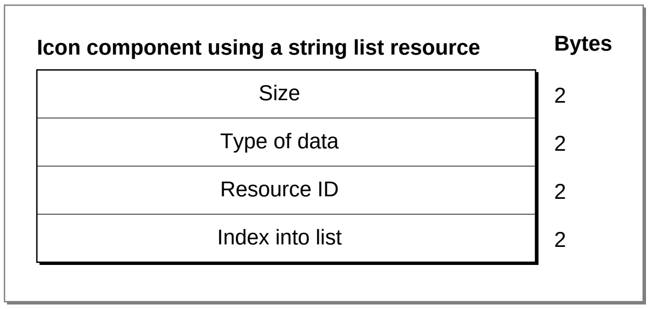


If you examine a compiled version of an `'hfdr'` resource, you find that a component identified in a Rez input file by the `HMStringItem` identifier consists of the following elements:

- 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

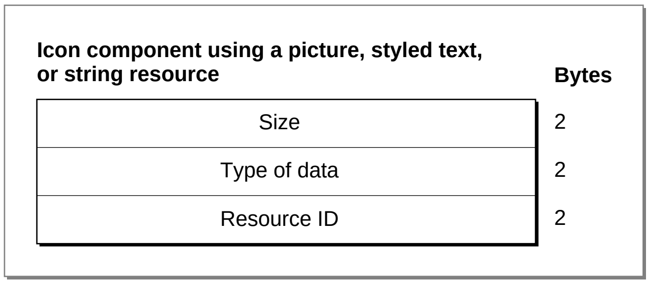


If you examine a compiled version of an `'hfdr'` resource, you find that a component identified in a Rez input file by the `HMStringResItem` identifier consists of the following elements:

- 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

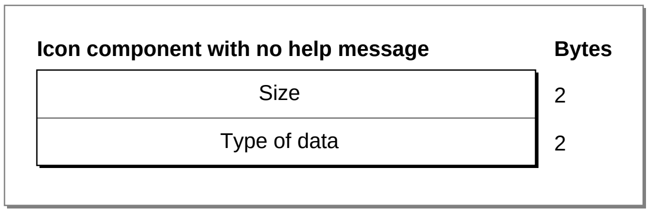
If you examine a compiled version of an `'hfdr'` resource, you find that a component identified in a Rez input file by either the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier consists of the following elements:

- 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

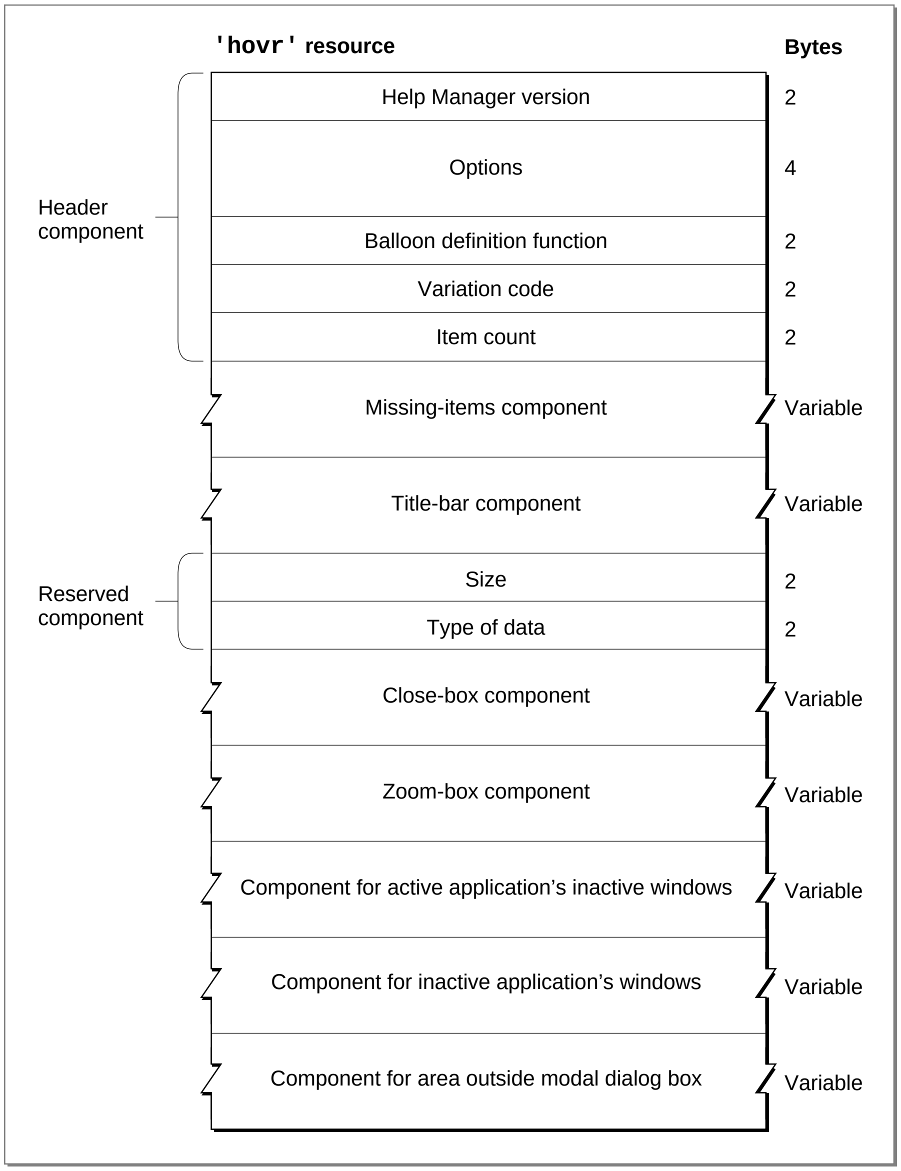
The Help Manager also provides default help balloons for the title bar and the close and zoom boxes of an active window, for the windows of inactive applications, for inactive windows of an active application, and for the area outside a modal dialog box.

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


'hovr' resource
Bytes
Header component
Help Manager version 2
Options 4
Balloon definition function 2
Variation code 2
Item count 2
Missing-items component Variable
Title-bar component Variable
Reserved component
Size 2
Type of data 2
Close-box component Variable
Zoom-box component Variable
Component for active application's inactive windows Variable
Component for inactive application's windows Variable
Component for area outside modal dialog box Variable

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.

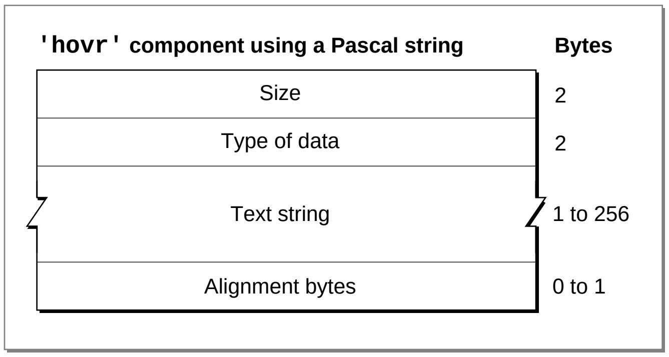
The structures of the remaining components depend on identifiers specified inside the components. The identifiers used in a Rez input file are described in "Specifying the Format for Help Messages" on page 3-23.

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

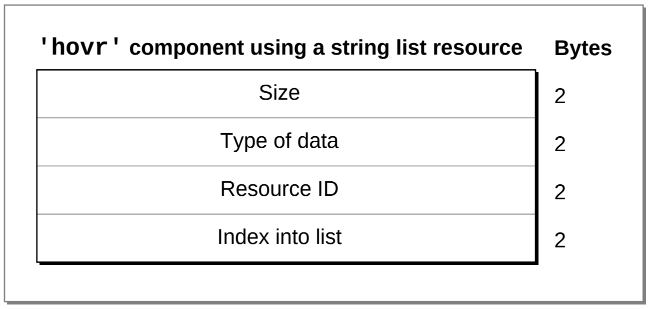


If you examine a compiled version of an `'hovr'` resource, you find that a component identified in a Rez input file by the `HMStringItem` identifier consists of the following elements:

- 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

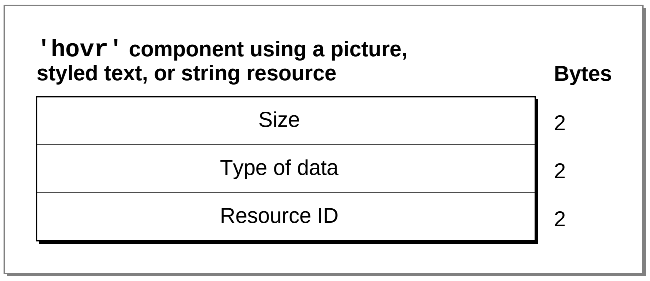


If you examine a compiled version of an `'hovr'` resource, you find that a component identified in a Rez input file by the `HMStringResItem` identifier consists of the following elements:

- 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

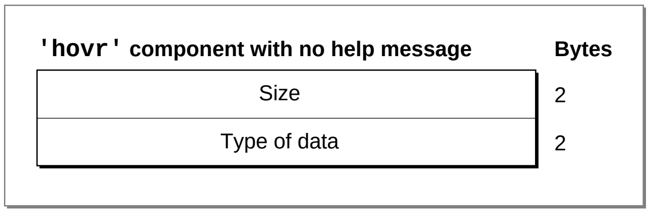
If you examine a compiled version of an `'hovr'` resource, you find that a component identified in a Rez input file by either the `HMPictItem`, `HMTEResItem`, or `HMSTRResItem` identifier consists of the following elements:

- 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