Help Balloons in Dynamic Windows

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

You should display or remove help balloons for dynamic windows at the same time that you normally check the mouse position to display or change the cursor. For example, if you provide your own DoIdle procedure, you can also check the mouse position and, if the cursor is over a hot rectangle, you should display the associated help balloon.

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

- identify the hot rectangles for each area or object
- create data structures to store the locations of the hot rectangles
- determine how to calculate their changing locations
- track and update the hot rectangles
- when the cursor is over a hot rectangle, display its help balloon by using the **HMShowBalloon** function

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

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

The **HMShowBalloon** function is useful for

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

For example, windows with scrolling file icons (such as Finder windows) or scrolling tool symbols (such as those shown in the Figure, "Static and Dynamic Windows" above) require you to use **HMShowBalloon** to display help balloons for the icons or symbols. Likewise, if you have tools-such as rulers that users configure for tab stops in a word-processing document-that scroll with a document, you will need to use **HMShowBalloon** to display help balloons for the scrolling tools.

The **Help Manager** provides default help balloons for certain areas of the window frame. **Overriding Other Default Help Balloons** describes how to override these default help balloons.

When using **HMShowBalloon**, you specify the help content, a tip location, a rectangle to use if the **Help Manager** needs to change the tip location, an optional pointer to a function that can modify the tip and rectangle coordinates, the balloon definition function, and the variation code. In the final parameter to the **HMShowBalloon** function, you should also provide a constant that tells the **Help Manager** whether it should save the bits behind the balloon.

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

Specify the help content in the aHelpMsg parameter to the **HMShowBalloon** function. You can specify the help information for each hot rectangle using text strings, 'STR ' resource types, 'STR#' resource types, styled text resources, 'PICT' resource types, handles to styled text records, or handles to pictures.

The **HMMessageRecord** data type defines the help message record.

The hmmHelpType field is a constant that specifies the data type of the next field of the help message record. The field following the hmmHelpType field can be one of a number of different data types. You specify the content of the help balloon in this field.

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

You specify a constant for the hmmHelpType field.

You can use the khmmString constant to specify a Pascal string. Here's an example of how to use the khmmString constant in the help message record. (Although you can specify a string from within your code, storing the strings in resources and then accessing them through the **Resource Manager** makes localization easier.)

```
#include <Balloons.h>
#include <string.h>
#include <pascal.h>

HMMessageRecord aHelpMsg;
```

```
Point   tip;
RectPtr     alternateRect;
Str255      hmmString;
OSErr  err;

aHelpMsg.hmmHelpType = khmmString;
strcpy (aHelpMsg.u.hmmString, "To turn the page, click here.");

// be sure to initialize tip and alternateRect here

CtoPstr (hmmString);
err = HMShowBalloon(&aHelpMsg,tip,alternateRect,
            nil,0,0,kHMRegularWindow);
```

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

Here's an example that specifies a 'PICT' resource ID.

```
#include <Balloons.h>

HMMessageRecord aHelpMsg;
Point           tip;
RectPtr             alternateRect;
OSErr           err;

aHelpMsg.hmmHelpType = khmmPict;
aHelpMsg.u.hmmPict = 128;                 // resource ID of 'PICT' resource

// be sure to initialize tip and alternateRect here

err = HMShowBalloon (&aHelpMsg, tip, alternateRect, nil, 0,
            0, kHMRegularWindow);
```

Here's an example of providing a handle to a 'PICT' resource.

```
#include <Balloons.h>

HMMessageRecord aHelpMsg;
PicHandle       pict;
Point           tip;
Rect            pictFrame;
RectPtr             alternateRect;
OSErr           err;

// be sure to initialize pictFrame here
pict = OpenPicture(&pictFrame);
```

```
DrawString("\pTest Balloon");
ClosePicture();
aHelpMsg.hmmHelpType = khmmPictHandle;
aHelpMsg.u.hmmPictHandle = (Handle)pict;

// be sure to initialize tip and alternateRect here

err = HMShowBalloon(&aHelpMsg, tip, alternateRect,
          nil, 0, 0, kHMRegularWindow);
KillPicture(pict);
```

The **HMStringResType** data type defines a **Help Manager** string list record.

The *hmmResID* field specifies the resource ID of the 'STR#' resource, and the *hmmIndex* field specifies the index of the string within that resource to use for the help information.

To use a string stored in an 'STR#' resource, use the khmmStringRes constant in the hmmHelpType field and use a record of data type **HMStringResType** in the next field.

```
#include <Balloons.h>

HMMessageRecord aHelpMsg;
Point   tip;
RectPtr     alternateRect;
OSErr  err;

aHelpMsg.hmmHelpType = khmmStringRes;
aHelpMsg.u.hmmStringRes.hmmResID = 1000;
aHelpMsg.u.hmmStringRes.hmmIndex = 1;

// be sure to initialize tip and alternateRect here

err = HMShowBalloon (&aHelpMsg, tip, alternateRect,
       nil, 0, 0, kHMRegularWindow);
```

To use styled text resources, use the khmmTERes constant in the hmmHelpType field. In the next field, supply a resource ID that is common to both a style scrap ('styl') resource and a 'TEXT' resource. For example, you might create a 'TEXT' resource with resource ID 1000 that contains the words "Displays your text in boldface print." You would also create an 'styl' resource with resource ID 1000 that applies boldface style to the word "boldface." When you specify the HMTEResItem constant and resource ID 1000 for a help balloon, the **Help Manager** employs **TextEdit** routines to display your text with your prescribed styles. (See **TextEdit** for a description of the style scrap.)

To use a handle to a styled text record, supply the khmmTEHandle constant in the hmmHelpType field.

```
#include <Balloons.h>

HMMessageRecord aHelpMsg;
Point           tip;
RectPtr         alternateRect;
TEHandle        hTE;
Rect            destRect,viewRect;
OSErr           err;

// be sure to initialize the destRect and viewRect
hTE = TEStylNew(&destRect,&viewRect); // or use TENew

// be sure to fill in data in handle here
aHelpMsg.hmmHelpType = khmmTEHandle;
aHelpMsg.u.hmmTEHandle = (Handle)hTE;

// be sure to initialize tip and alternateRect here
err = HMShowBalloon (&aHelpMsg, tip, alternateRect,
            nil, 0, 0, kHMRegularWindow);
```

You specify the tip and the rectangle pointed to by alternateRect in global coordinates. The **Help Manager** calculates the location and size of the help balloon. If the help balloon fits on screen, the **Help Manager** displays the help balloon using the specified tip.

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

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 on screen; if you specify an alternate rectangle that is larger than your hot rectangle, you have greater assurance that the help balloon will not obscure some element explained by the balloon.

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

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

- the help balloon's tip is offscreen or in the menu bar and you do not specify an alternate rectangle

- both the help balloon's tip and the alternate rectangle are offscreen

The final parameter in **HMShowBalloon** specifies how the **Help Manager** should draw and remove the balloon. Use the constants shown in the section called, **Help Manager Data** for the parameter.

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-and 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 only use this 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 popup 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 will rarely need this option. It is necessary only in a modal environment that might immediately change to a nonmodal environment-that is, where the bits behind the help balloon are static when the balloon is drawn, but can possibly change before the help balloon is removed.

The Listing below shows a sample routine that displays help balloons for hot rectangles within the content area of a window.

Using HMShowBalloon to display help balloons

```
// Assuming inclusion of <MacHeaders>

#include <Balloons.h>
#include <THINK.h>

#define OurHelpMsgsID   128;    // The 'STR#' resource ID

void FindAndShowBalloon (WindowPtr window);

Rect    gMyPredefinedRects[10];
short   gLastBalloon;

void FindAndShowBalloon (WindowPtr window)
{
    short           i;
    Point           mouse;
    GrafPtr         savePort;
    HMMessageRecord helpMsg;
    OSErr           result;
    Boolean         inRect;
    Rect            hotRect;


    if (window == FrontWindow() ) {// Only do frontmost windows

        GetPort(&savePort);     // Save the old port for later
        SetPort(window);    // Set the port to the front window
        GetMouse(&mouse);// Get the mouse in local coords
        inRect = FALSE;    // Clear flag saying mouse wasn't in any Rect

        if (PtInRect(mouse, &(window->portRect)) )

            // If the cursor is in the window
            for ( i = 0; i < 10; i++)   // Check all 10 predefined
                                        // rects in the window

                if ( PtInRect(mouse, &gMyPredefinedRects[i]) ) {

                    // Cursor in rect

                    if (i != gLastBalloon) {

                    // Cursor wasn't same as last time

                        hotRect = gMyPredefinedRects[i];
                        LocalToGlobal(&(topLeft(hotRect)));

                        // Converting rect to global

                        LocalToGlobal(&(botRight(hotRect)));

                        // Put the tip in the middle

                        SetPt(&mouse, (hotRect.left - hotRect.right) / 2,
                            (hotRect.bottom - hotRect.top) / 2);
                        helpMsg.hmmHelpType = khmmStringRes;
                        // Want 'STR#' resource

                        helpMsg.u.hmmStringRes.hmmResID = OurHelpMsgsID;

                        // This resID

                        helpMsg.u.hmmStringRes.hmmIndex = i+1; // This index

                        result = HMShowBalloon(&helpMsg,
                                                        // Use just-made help msg
                                          mouse, // Pointing to this tip
                      (RectPtr)&gMyPredefinedRects[i], // This rectangle
                                              nil,     // No special tip proc
                                            0,0,    // Using default balloon
                                          FALSE);    // Don't save bits behind

                        if (result)     // Then remember balloon
                        gLastBalloon = i;
                      }
                      inRect = TRUE;     // Remember when the cursor is in
                          // any rect
                    }
                if (!inRect)
                    gLastBalloon = -1;     // Clear last balloon global for no hit
                SetPort(savePort); // Restore the port
            }
        }
```

The FindAndShowBalloon procedure in the Listing above tracks the cursor, and, if the cursor is over a predefined hot rectangle, it displays a help balloon for that rectangle. In this example there are 10 predefined rectangles (in the MyPredefinedRects array) and 10 corresponding help messages in an 'STR#' resource (of ID OurHelpMsgsID)-one message for each hot rectangle. Other supporting routines can update the coordinates of the hot rectangles as their locations change.

You can also use the **HMShowBalloon** function from the filter function of a modal dialog box or alert box.

For more information on the **HMShowBalloon** and **HMRemoveBalloon** functions see, **Displaying and Removing Help Balloons** .