Popup CDEF v1.0b4

Contents

Description

This is version 1.0b4 of Popup CDEF. It now works with color graphics ports and supports colorized menus.

This CDEF implements a popup menu control. The CDEF handles display of the menu's title, the current selection, the one pixel drop shadow, and the down arrow at the end of the menu. It also handles tracking of the mouse and checking and unchecking of the current item. It is compatible with systems 6.0.5 and 7.0.

The CDEF is modeled after the popup CDEF provided by Apple in System 7.0 and described in IM-VI, p3-16 to 3-19. If you are already familiar with that CDEF then using this CDEF will be very simple. Additional support is provided for type-in popup menus, for menus created dynamically by the application, and for colorized menus.

This program is free, and can be used subject to the terms detailed in the file "Distribution".

Using the CDEF

This section describes how you can use the popup CDEF in your own applications. This section was adapted from IM-VI, p3-16 to 3-19.

Adding the CDEF to your Application

You'll need to use ResEdit, or some other resource editing utility, to copy the 'CDEF' resource with ID 129 from the file "PopupCDEF" to your application. The file "PopupCDEF.h" contains definitions of constants and types you can use to create and access the popup CDEF. The constant kPopupProcID is defined as 2064 in the file "PopupCDEF.h". The procID code to pass to NewControl is computed by multiplying the CDEF's resource ID by 16, and then adding any variation codes. For instance, the basic popup CDEF would have a procID of 16*129 = 2064. A popup control with the popupUseWFont variation code would have a procID of 16*129 + 8 = 2072. If you already have a CDEF with the same ID in your application, then you can change the ID of the popup CDEF resource, but you'll have to calculate the procID codes for your controls using the new resource ID.

Creating and Using a Popup Menu Control

You can use NewControl to create a popup menu control, or you can use a 'CNTL' resource and allow the Dialog Manager to create the popup control in your dialogs. To create a control using the CDEF, you should use a procID of kPopupProcID (unless you changed the ID of the CDEF, as described above), and then add any desired variation codes. The value, min, max, and refCon parameters to NewControl have special meanings when creating a popup menu control, and are described in the following sections.

When editing dialogs with ResEdit, you may need to modify a control item's rectangle in the dialog item list resource. ResEdit doesn't always update the rectangle in the dialog item list resource (the 'DITL') to match the width and height specified in the 'CNTL' resource for the dialog item. If you click on a popup menu item, and the menu doesn't pop up, it is possible that the rectangle in the 'DITL' is not correct. To fix the rectangle in the 'DITL', you can open the 'DITL' resource with the 'DITL' template (not by double clicking on the 'DITL' resource), then set the rectangle of the item to the width and height specified in the 'CNTL' resource.

If you use the dialog manager, tracking of the mouse over the popup menu will be handled for you by DialogSelect. If you create a popup menu control in a regular window, then when the user clicks in a popup menu control tou must call the Control Manager routine FindControl, which will report a click in the control. You should then call TrackControl with an actionProc parameter of -1. The CDEF will hilite the control's title and will call PopupMenuSelect to display the popup menu and allow the user to select an item from the menu. Once the user has selected an item, the CDEF will set the control's value to the selected item number and will place a mark next to the selected item.

You can use the Control Manager routines GetCtlValue to determine which menu item is currently selected, and SetCtlValue to set the currently selected menu item. You can determine the number of items in the menu using GetCtlMax. You should only SetCtlMax if you change the number of items in the menu. You should never call SetCtlMin, as the minimum value of the control must always be set to 1.

The file "PopupDemo.c" contains the source code to an application that demonstrates the different types of popup menus that can be created with this CDEF. The file also contains some code snippets that illustrate how you can support type-in popup menus in your own applications.

Value Parameter

The value parameter specifies the style in which to draw the popup's title. The value parameter can be any combination of the following values (defined in <Controls.h>):

popupTitleLeftJust	0x0000 left aligned text
popupTitleCenterJust	0x0001 ignored
popupTitleRightJust	0x00FF right aligned text
popupTitleBold	0x0100 bold text
popupTitleItalic	0x0200 italic text
popupTitleUnderline	0x0400 underlined text
popupTitleOutline	0x0800 outlined text
popupTitleShadow	0x1000 shadow text
popupTitleCondense	0x2000 condensed text
popupTitleExtend	0x4000 extended text
popupTitleNoStyle	0x8000 unstyled text

If you use popupTitleRightJust, then the popup's title is drawn to the right of the popup box instead of to its left. After the control has been created, the value parameter is used to determine the currently selected item. Initially, the first item in the menu is selected.

Min Parameter

The min parameter specifies the resource ID of the menu in the popup menu control. The CDEF calls GetResource('MENU', id), where id is the value of the min paremeter, to determine if the menu has already been loaded by GetMenu. If it has been loaded by GetMenu, then the handle returned by GetResource is used for the control's menu handle. Otherwise, the CDEF calls GetMenu(id) to load the menu and uses the handle returned by GetMenu for the control's menu handle. If, however, the menu is not in a resource file, then the CDEF calls GetMHandle(id), and uses the handle returned as the control's menu handle. This allows the application to dynamically create a menu for use by the control, though the application must insert the menu into the menu list before the control is created. If the CDEF used GetMenu to load the menu handle, then it uses ReleaseResource when the control is disposed of to release the memory used by the menu handle. Otherwise, it is the application's responsibility to release the memory used by the menu handle. After the control has been created, the min parameter is set to 1.

Max Parameter

The max parameter determines the width of the control's title area. Normally, you'll pass zero for the max parameter, allowing the title area to be resized dynamically to fit the title string. If you require a fixed width for the title, you can use the max parameter to set its width. If the max parameter is less than the minimum width of the title area, then the max parameter is ignored and the minimum width is used instead. The minimum width depends on the size of the font and the font style used to display the title. After the control has been created, the max parameter is set to the number of items in the popup menu.

Variation Codes

The procID parameter should contain the resource ID of the popup CDEF multiplied by 16, plus any desired variation code. The variation code can be any combination of the following:

popupFixedWidth	0x0001 fixed-width control
popupTypeIn	0x0002 type-in style popup menu
popupUseAddResMenu	0x0004 use AddResMenu and refCon parameter
popupUseWFont	0x0008 use window's font and font size

popupFixedWidth

PopupFixedWidth makes the control fill the entire control rectangle assigned to it. Without this variation code, the control is resized dynamically to take up only as much space as is needed to display the menu and its title. PopupFixedWidth is especially useful when you want to align the left and right edges of several popup menus. When used in combination with a fixed width for the control titles (specified in the "max" parameter to NewControl), you can arrange several popup menus to be aligned at both their left and right edges and to have their titles aligned. This often results in a more aesthetically pleasing arrangement for the controls.

popupTypeIn

The popupTypeIn variation code is used for type-in popup menus, as described in IM-VI, p2-37. With this variation code, only the down arrow and the one

pixel border and drop shadow are drawn. The control's title and current selection are not drawn. The application must hilite and unhilite the type-in field and must handle insertion and deletion of the menu item containing a value not already in the popup menu. The popupTypeIn variation code is defined in the file PopupCDEF.h.

popupUseAddResMenu

The popupUseAddResMenu variation code causes the CDEF to insert items into the menu using the AddResMenu procedure. The CDEF interprets the refCon parameter as a value of type ResType that specifies the resource type to load. For instance, to provide a popup menu listing available fonts, you would use the variation code popupUseAddResMenu and set the refCon parameter to 'FONT'. The control's refCon field is available for the application's use once the control has been created, or if the popupUseAddResMenu variation code isn't used.

popupUseWFont

Uses the font of the window containing the control instead of the system font. The control's title and current selection are drawn using the font and size of the window. The menu, when selected by the user, is also drawn in the window's font and font size.

Disabling a Menu Item

The current selection rectangle is displayed in a grayed out pattern when the menu item corresponding to the currently selected item is disabled. If you disable (or enable) the menu item after the current item has been selected, then you'll need to redraw the control by using InvalRect on the control's rectangle to generate an activate event. You can also redraw the control by using any Control Manager routine that will redraw the control. The entire control is drawn in a grayed out pattern when the control is disabled by setting its hilite value to 255 with HiliteControl.

Colorizing a Popup Menu

The popup menu control will be drawn in color if there is a 'mctb' resource with the same ID as the menu. The color entries from the 'mctb' resource are applied to the parts of the popup menu. In a future version, I will also add support for a control color table.

If the selected menu item has an icon associated with it, then the CDEF first looks for a 'cicn' resource before looking for an 'ICON' or 'SICN' resource. If a 'cicn' resource is found, then it is drawn in the current selection box. Otherwise, the 'SICN' or 'ICON' resource is used.

On Macintosh models that do not support Color QuickDraw (such as the Plus and SE), the CDEF will ignore any 'mctb' and 'cicn' resources.

Accessing the Menu Handle

You can access the menu handle and menu ID of the menu associated with the control by dereferencing the contrlData field of the control record. The contrlData field is a handle to a block of private information used by the CDEF. The first few bytes of this block contain the following structure:

```
/* This is the same structure as the popupPrivateData record described in IM-VI, p3-19. By making this the first
element in the popup structure, a CDEF based on this popup library is made more compatible with Apple's
CDEF. */
typedef struct {
    MenuHandle mHandle;
    short          mID;
} PopupPrivateType, *PopupPrivatePtr, **PopupPrivateHandle;
```

This structure is defined in the file PopupCDEF.h.

For instance, you could use the following function to access the menu handle of a popup menu control:

```
/* return a handle to the popup control's menu */
MenuHandle GetCtlMenu(ControlHandle ctl)
{
    return((**(PopupPrivateHandle) (**ctl).contrlData).mHandle);
}
```

To access the menu handle of a popup menu control item in a dialog, you could use the following functions:

```
/* return the control handle for the dialog item */
ControlHandle GetDControl(DialogPtr dlg, short item)
{
    short type;
    Handle hitem;
    Rect box;

    GetDItem(dlg, item, &type, &hitem, &box);
    return((ControlHandle) hitem);
}

/* return the menu handle for the popup control item */
MenuHandle GetDMenu(DialogPtr dlg, short item)
{
    return(GetCtlMenu(GetDControl(dlg, item)));
}
```

Differences from the System CDEF

This section is intended for people who are already using the popup CDEF provided with System or who are trying to decide which popup CDEF to use. While this popup CDEF is similar to the popup CDEF provided with System, the two CDEFs are not identical. Following is a list of the most important differences between the two CDEFs.

Extensions to the System CDEF

An additional variation code is defined, popupTypeIn. With this variation code, only the popup's down arrow is drawn, the popup's title and current selection are not drawn. You can use this variation code to support type-in popup menus. Type-in popup menus are described in IM-VI, p2-37.

Color menus are supported. Popup menus are colorized as specified by a menu's color table resource. The popup menu control will be drawn in color if there is a 'mctb' resource with the same ID as the menu. If the selected menu item has an icon associated with it, then the CDEF first looks for a 'cicn' resource before

looking for an 'ICON' or 'SICN' resource. If a 'cicn' resource is found, then it is drawn in the current selection box.

To support menus created dynamically by the application, if the menu isn't in the application's resource file, then the menu is searched for in the menu list using GetMHandle. For this feature to work, the menu must not be in any resource file open at the time the popup control is created, and the menu must be inserted into the menu list before the popup control is created.

When using the same menu in more than one popup menu control, the currently selected item is marked correctly in each popup menu control when the popup menu is displayed. In the System CDEF, if popup menu controls share the same menu, then multiple items may be marked in each of the menus; this is confusing to the user, who should see only one marked item at any time.

Features in the System CDEF that are not Supported

The following features of the System popup CDEF are not supported by this CDEF:

popupTitleCenterJust	ignored

Other Differences

The System popup CDEF will only display the control's title if a specific title width is specified (in the "max" parameter to NewControl) when the control is created. My popup CDEF will always show the title, unless the control has no title or the popupTypeIn variation code is used. Specifying a specific width for the title still works for my popup CDEF, and can be used when you want to align several popup menus.

The appearance of popup menus drawn with this CDEF varies slightly from the appearance of popup menus drawn with the System popup CDEF. Except for the differences mentioned above, the variations are minor, and shouldn't be noticed by most users.

Bugs

This section lists significant bugs that must fixed in the popup CDEF. The items are listed in approximate order of urgency. Most of these things need to be fixed in the file "PopupLib.c".

(There are no known bugs.)

Things To Do

This section lists things that should be fixed in, or added to, the popup CDEF. The items are listed in approximate order of urgency. Most of these things need to be fixed in the file "PopupLib.c".

Support for a control color table needs to be added, so that the different parts of the control can be colorized (such as the control's frame). When there's a conflict between a control's color table and a menu's color table, I haven't

decided whether the control's color table should override the conflicting entries in the menu's color table, or vice-versa.

There are some slight differences between where the popup CDEF draws the text of the current selection and where the MDEF draws the text of the current selection. If I could figure out how the MDEF calculates the text position then I could emulate its behavior.

Some basic configurations have not been tested, since I don't have access to the appropriate hardware and/or software. The following configurations need to be tested:

- a Macintosh with System 6 and a color monitor;
- a Macintosh with System 6 and multiple monitors, each with different pixel depths;
- a non-Roman script system;
- a right-to-left script system, such as Hebrew or Arabic;
- mixed or bidirectional script systems, such as Roman and Hebrew;
- multibyte script systems, such as Kanji.

There are additional notes, marked with "program_note" in the source files, of things that I wasn't able to implement or fix for one reason or another.

A minor alignment problem may occur when the text style of the popup's title results in a line height different from the line height of the plain system font (or, if the wfont attribute is true, then different from the height of the window's font). For instance, if the title's style is outline+underline+bold, its height may differ by several pixels from the height currently calculated.

Credits

Eric Bowman (bobo@reed.edu) suggested and helped me add a call to GetMHandle to allow use of menus created by the application that are not in the resource file. He also pointed out the use of a handle that had already been disposed of.

Leonard Roesenthal (leonardr@netcom.com) and Stephen Croot (Stephen_Croot@macconn.mpx.com.au) explained how to force the width of the popup menu to cover the down arrow when PopupMenuSelect is called.

Stefan Kurth (stk@contrib.de) and Dave Nebinger (dnebing@andy.bgsu.edu) helped me figure out how to draw disabled menu items in color.

Peter Lewis (peter@ncrpda.curtin.edu.au) found a bug in the function MenuItemEnabled that would prevent the use of menu items after the 15th item.

How to Contact Me

You can reach me at the following address:

Ari Halberstadt

9 Whittemore Road
Newton, MA 02158-2105
USA

ari@world.std.com