Popup CDEF v1.0b5

Contents

Description

This is version 1.0b5 of Popup CDEF. It now works with color graphics ports and supports color menus and control color tables.

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 triangle 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 "Inside Macintosh: Toolbox Essentials"[1]. If you are already familiar with the system 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 color menus and controls.

This program is free, and can be used subject to the terms detailed in the file "Distribution". all source code in C and a Symantec C v7.0 project files are included.

Using the CDEF

This section describes how you can use the popup CDEF in your own applications. Parts of this section were adapted from Inside Macintosh, Volume VI, p3-16 to 3-19.

Adding the CDEF to Your Program

You must add the 'CDEF' resource with ID 129 to your program's resource fork. You can use ResEdit, or a similar resource editing utility, to copy the resource from the file "PopupCDEF.rsrc" to your program. When using Symantec's Project Manager, I prefer to add the file "PopupCDEF.rsrc" to my project file, allowing the project manager to automatically copy the resources. In MPW, I prefer to add a line to my makefiles to use the MPW tool Rez to copy the resource. Using CodeWarrior DR4, you must copy the resource into a file named "Project.rsrc", where "Project" is the name of your project file.

Resource and Procedure ID

If you already have a 'CDEF' resource in your application with ID 129, then you will need to change the ID of the popup CDEF resource. You can use any resource editing utility to change the resource's ID. The resource can be assigned any ID greater than 127.

The resource ID of the 'CDEF' resource is multiplied by 16 to get the base ID for the procID parameter to NewControl. The constant kPopupProcID is defined in the file "PopupCDEF.h" as 2064, which is 16*129. If you changed the resource ID of the 'CDEF' resource, then you will need to use a different constant to specify the procID. For instance, if you changed the resource's ID to 200, you would need to use the value 3200 as the procID. All the examples in this documentation assume you are using the constant kPopupProcID, but you can substitute your own constant if necessary.

[1]See also Inside Macintosh, Volume VI, p3-16 to 3-19.

Creating and Using a Popup Menu Control

Examples

The file "PopupDemo.c" contains the source code to an application that demonstrates some of the variations of popup menu controls 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. The files "PopupDemo.r" and "PopupDemoIcons.r" contain the resources used by the application.

Using Popup Constants in Source Code Files

You need to include the file "PopupCDEF.h" to access any of the constants that start with "kPopup". These constants include kPopupProcID and the part identifiers used for colorizing a popup menu control using a control color table. To use this file in a resource definition file, such as a Rez ".r" file, you must define the constant POPUP_CDEF_RESOURCE_ONLY as 1 before including "PopupCDEF.h". For instance, you could use the following lines to include the file in your resource definition file.

```
#define POPUP_CDEF_RESOURCE_ONLY (1)
#include "PopupCDEF.h"
... resource definitions ...
```

Additional popup menu control constants and types are defined in the file <Types.r>.

You may also want to include "PopupCDEF.h" in your source code files (e.g., ".c" files). You can simply include the file using an include preprocessor directive, as you do not need to use any special defines before including the file.

Creating a New 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, 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 does not 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.

Handling Clicks

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 you 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.

Getting and Setting the Current Item

You can use the Control Manager routines GetControlValue and SetControlValue to determine which menu item is currently selected. You can determine the number of items in the menu using GetControlMaximum. The control's minimum value is automatically set to 1 if there are any items in the menu, otherwise it is set to 0. The control's maximum value is automatically set to the number of items in the menu. The control's value is always clipped to the control's minimum and maximum values.

Value Parameter

The value parameter specifies the style in which to draw the popup's title. The value parameter can be any combination one of the following values.

popupTitleLeftJust	left aligned text
popupTitleCenterJust	ignored
popupTitleRightJust	right aligned text

If you use popupTitleRightJust, then the popup's title is drawn to the right of the popup box instead of to its left.

Any one of the above values can be combined with any number of the following values.

popupTitleBold	bold text
popupTitleItalic	italic text
popupTitleUnderline	underlined text
popupTitleOutline	outlined text
popupTitleShadow	shadowed text
popupTitleCondense	condensed text
popupTitleExtend	extended text
popupTitleNoStyle	unstyled text

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, unless the menu is empty, in which case no item 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 parameter, 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, or to 0 if the menu is empty.

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 control
popupUseAddResMenu	0x0004 use AddResMenu and refCon parameter
popupUseWFont	0x0008 use window's font and font size

popupFixedWidth

The popupFixedWidth variation code 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. With this variation code, only the triangle 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

The popupUseWFont variation code causes the CDEF to use 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. Finally, the character used to mark the selected menu item is set to '•' (bullet) rather than '√' (checkmark).

Adding Color to a Popup Menu Control

A popup menu control will be drawn in color if it is drawn into a color graphics port. The parts of a control that can be colorized are identified by constants defined in the header "PopupCDEF.h". The part identifiers are described in the following table.

kPopupPartTitleFore	Specifies the color in which the text of the control's title is drawn if the control is enabled. If the control is disabled, the text is drawn in a color derived from a blend of the background and foreground colors.

kPopupPartTitleBack	Specifies the color in which the background of the control's title rectangle is drawn.
kPopupPartTitleDisabled	Specifies the color in which the text of the title is drawn if the control is disabled.
kPopupPartItemFore	Specifies the color in which the text of the control's currently selected item is drawn if the item and the control are enabled. If either the item or the control are disabled, the text is drawn in a color derived from a blend of the background and foreground colors.
kPopupPartItemBack	Specifies the color in which the background of the item's rectangle is drawn.
kPopupPartItemDisabled	Specifies the color in which the text of the item is drawn if either the item or the control is disabled.
kPopupPartIconFore	Specifies the color in which the foreground of the black and white icon associated with the control's currently selected item is drawn. A black and white icon can be a 'SICN' or an 'ICON' resource. A color icon 'cicn' resource is used in preference to a black and white icon if a 'cicn' resource with the same resource ID exists. If either the item or the control are disabled, the icon is drawn in a color derived from a blend of the background and foreground colors.
kPopupPartIconBack	Specifies the color in which the background of the icon is drawn.
kPopupPartIconDisabled	Specifies the color in which the foreground of the icon is drawn if either the item or the control is disabled.
kPopupPartTriangleFore	Specifies the color in which the triangle is drawn. If the control is disabled, the triangle is drawn in a color derived from a blend of the background and foreground colors.
kPopupPartTriangleBack	Specifies the background color in which the triangle is drawn.
kPopupPartTriangleDisabled	Specifies the color in which the triangle is drawn if the control is disabled.
kPopupPartFrameFore	Specifies the color in which the frame around the current selection is drawn. If the control is disabled, the frame is drawn in a color derived from a blend of the background and foreground colors.
kPopupPartFrameBack	Specifies the background color in which the frame is drawn.
kPopupPartFrameDisabled	Specifies the color in which the frame is drawn if the control is disabled.
kPopupPartShadowFore	Specifies the color in which the shadow of the frame around the current selection is drawn. If the control is disabled, the shadow is drawn in a color derived from a blend of the background and foreground colors.
kPopupPartShadowBack	Specifies the background color in which the shadow is drawn.
kPopupPartShadowDisabled	Specifies the color in which the shadow is drawn if the control is disabled.

Note: need a picture.

You can add specific colors to the parts of a popup menu control by attaching a menu color table to the popup control's menu, or by attaching a control color table to the popup control. So that all popup menu controls have a consistent appearance across applications, you should generally refrain from colorizing the controls. If you need to add color to a control, you should use a menu color table, since this will give the most consistent appearance between the control and the menu displayed when the user clicks on the control. Finally, if you need to add color to the parts of a control that can not be colorized using a menu color table—such as the triangle, frame, and shadow—you can use a control color table.

Color Mapping

There are five methods by which the color of each part of a control can be specified. The methods that can be used, and the order in which they are applied: menu color table (mct), control color table (cct), default menu color table (dmct), default control color table (dcct), and default. The following table shows the specific mapping order used for each part of a control.

Part	mct	cct?	dmct?	dcct?	default	
kPopupPartTitleFore	mct	cct	dmct	dcct	black	
kPopupPartTitleBack	mct	cct	dmct	dcct	white	
kPopupPartTitleDisabled	—		cct	—	dcct	gray
kPopupPartItemFore	mct	cct	dmct	dcct	black	
kPopupPartItemBack	mct	cct	dmct	dcct	white	
kPopupPartItemDisabled	—		cct	—	dcct	gray
kPopupPartIconFore	mct	cct	dmct	dcct	black	
kPopupPartIconBack	mct	cct	dmct	dcct	white	
kPopupPartIconDisabled	—		cct	—	dcct	gray
kPopupPartTriangleFore	—		cct	—	dcct	black
kPopupPartTriangleBack	—		cct	—	dcct	white
kPopupPartTriangleDisabled	—		cct	—	dcct	gray
kPopupPartFrameFore	—		cct	—	dcct	black
kPopupPartFrameBack	—		cct	—	dcct	white
kPopupPartFrameDisabled	—		cct	—	dcct	gray
kPopupPartShadowFore	—		cct	—	dcct	black
kPopupPartShadowBack	—		cct	—	dcct	white
kPopupPartShadowDisabled	—		cct	—	dcct	gray

Once the foreground and background colors have been determined, if part of a control is disabled, and if a kPopupPart…Disabled part was not found in the control's color table, then the part's foreground and background colors are blended to form a new foreground color. For instance, a blend of a black foreground color and a white background color would result in a gray foreground color (the background color is unaffected).

For example, to determine the colors to use for a control's title, the CDEF first looks for a menu color table entry. If the menu color table entry is found, then the colors specified in the entry are used for the title's foreground and background. If a menu color table entry is not found, then the CDEF looks for a color table for the control with entries for each of the kPopupPartTitleFore and kPopupPartTitleBack part identifiers. If either part identifier does not have a corresponding entry in the control's color table (or if the control does not have a color table), then the CDEF looks in the default menu color table. If the default menu color table does not exist, then the CDEF looks in the default control color table for entries with part identifiers cTextColor and cBodyColor. If the default color table does not contain entries for these part identifiers (or if the default color table does not exist) then the CDEF uses black for the title's foreground color and white for the title's background color. Finally, if the control is disabled, and if the control's color table contains an entry for the kPopupPartTitleDisabled part identifier, then that color is used as the foreground color in which the title string is drawn, otherwise a color derived from a blend of the foreground and background colors is used for the title string (the color is actually derived using the grayishTextOr transfer mode).

The following sections explain each method of color mapping in greater detail.

Menu Color Table

To add a menu color table to a popup menu control, you can either add a 'mctb' resource with the same ID as the menu's ID, or you can use the SetMCInfo routine to add a color table to an existing menu. You can find more information about menu color tables in "NIM:Toolbox Essentials".

A menu color table contains three types of entries: an entry for the menu's title, a default entry for all items for which no color is specified, and entries for each item in a menu. Each entry in a menu color table has four RGB values, called mctRGB1, mctRGB2, mctRGB3, and mctRGB4. The entries in a menu color table are mapped to the part identifiers of the popup CDEF as shown in the following table.

kPopupPartTitleFore	mctRGB1 of item 0
kPopupPartTitleBack	mctRGB4 of item 0
kPopupPartTitleDisabled	not mapped
kPopupPartItemFore	if item has a color entry, mctRGB2 of that item, otherwise mctRGB3 of item 0
kPopupPartItemBack	if item has a color entry, mctRGB4 of that item, otherwise mctRGB4 of item 0
kPopupPartItemDisabled	not mapped
kPopupPartIconFore	if item has a color entry, mctRGB42 of that item, otherwise mctRGB3 of item 0
kPopupPartIconBack	if item has a color entry, mctRGB4 of that item, otherwise mctRGB4 of item 0
kPopupPartIconDisabled	not mapped
kPopupPartTriangleFore	not mapped
kPopupPartTriangleBack	not mapped
kPopupPartTriangleDisabled	not mapped
kPopupPartFrameFore	not mapped
kPopupPartFrameBack	not mapped
kPopupPartFrameDisabled	not mapped
kPopupPartShadowFore	not mapped
kPopupPartShadowBack	not mapped
kPopupPartShadowDisabled	not mapped

Default Menu Color Table

The default menu color table entry is the menu color entry for the menu bar. A menu bar color table entry contains four entries, each corresponding to part of a menu. You can find more information about menu color tables in "Inside Macintosh:Toolbox Essentials".

The menu bar color entries are mapped to the parts of a popup menu control as shown in the following table.

kPopupPartTitleFore	mctRGB1
kPopupPartTitleBack	mctRGB4
kPopupPartTitleDisabled	not mapped
kPopupPartItemFore	mctRGB3
kPopupPartItemBack	mctRGB2
kPopupPartItemDisabled	not mapped
kPopupPartIconFore	mctRGB3
kPopupPartIconBack	mctRGB2
kPopupPartIconDisabled	not mapped
kPopupPartTriangleFore	not mapped
kPopupPartTriangleBack	not mapped
kPopupPartTriangleDisabled	not mapped
kPopupPartFrameFore	not mapped
kPopupPartFrameBack	not mapped
kPopupPartFrameDisabled	not mapped
kPopupPartShadowFore	not mapped
kPopupPartShadowBack	not mapped
kPopupPartShadowDisabled	not mapped

Control Color Table

To attach a control color table, you can either add a 'cctb' resource with the same ID as the 'CNTL' resource's ID, or you can use the SetControlColor routine to add a color table to an existing control. You can find more information about control color tables in "NIM:Toolbox Essentials".

A control color table may contain an arbitrary number of entries. Each entry contains the part identifier to which the entry applies and the RGB color of the part. The entries do not have to be in any specific order. The entries of a control color table for a popup menu control must use the part identifiers declared in "PopupCDEF.h". The color for each entry is used to draw the corresponding part of the control.

Default Control Color Table

The values specified by the default color table for push button, check box, and radio button controls are used if no entry for a control's part is found in a menu or control color table. Since the default color table provides only three part identifiers—cTextColor, cBodyColor, and cFrameColor—the part identifiers are mapped to the part identifiers of the popup CDEF as shown in the following table.

kPopupPartTitleFore	cTextColor
kPopupPartTitleBack	cBodyColor
kPopupPartTitleDisabled	not mapped

kPopupPartItemFore	cTextColor
kPopupPartItemBack	cBodyColor
kPopupPartItemDisabled	not mapped
kPopupPartIconFore	cTextColor
kPopupPartIconBack	cBodyColor
kPopupPartIconDisabled	not mapped
kPopupPartTriangleFore	cTextColor
kPopupPartTriangleBack	cBodyColor
kPopupPartTriangleDisabled	not mapped
kPopupPartFrameFore	cFrameColor
kPopupPartFrameBack	not mapped
kPopupPartFrameDisabled	not mapped
kPopupPartShadowFore	cFrameColor
kPopupPartShadowBack	not mapped
kPopupPartShadowDisabled	not mapped

By mapping the default table to a popup menu control's table, the CDEF supports a consistent color scheme on systems on which users have customized the default table.

Default Colors

The CDEF uses default colors if no specific color information is provided by the application or when drawing into a black and white graphics port. In these situations, the foreground color for enabled parts defaults to black and the background color defaults to white. The foreground color for disabled parts in a color graphics port defaults to gray.

Color Icons

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.

Black and White Computers

On Macintosh models that do not support Color QuickDraw (such as the Plus and SE), the CDEF ignores any menu or control tables, control color tables, 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:

```
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 a popup menu control's menu */
MenuHandle GetControlMenu(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 GetDialogItemControl(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 menu control item */
MenuHandle GetDialogItemMenu(DialogPtr dlg, short item)
{
   return(GetControlMenu(GetDialogItemControl(dlg, item)));
}
```

Dynamically Altering a Menu

It is possible, and often necessary, to use the Menu Manager to dynamically alter the appearance of a menu that is used in a popup menu control. For instance, you may want to change the text of an item, or you may want to enable or disable an item. Unfortunately, there is no direct communication between the Menu Manager and the Control Manager. If the item you are changing is also the currently selected item in the popup menu control, then you will have to make additional calls to the Control Manager to get the popup menu control to update the display of the menu item. The following routine can be used to force the popup menu control to update its display.

```
/* call after changing a popup menu's control menu */
void ChangedControlMenuItem(ControlHandle ctl)
{
   Rect contrlRect;

   GetPort(&port);
   SetPort((**ctl).contrlOwner);
   contrlRect= (**ctl).contrlRect;
   InvalRect(&contrlRect);
   SetPort(port);
}
```

This routine simply invalidates the rectangle containing the popup menu control. If you want the control to be updated immediately, you can replace the inside of the function with DrawControl. Whatever you make the body of ChangedControlMenu, I suggest you call this routine, rather than calling InvalRect or DrawControl directly, whenever you alter a menu item or a menu's color table in a menu used by a popup menu control. You should also call this function if you change the color table of a menu used by a popup menu control, and if you change the control color table of a popup menu control.

The next two subsections show how the ChangedControlMenu routine can be used. You can create similar routines to support additional modifications to menus.

Setting the Text of a Menu Item

You can use the ChangedControlMenu routine when setting the text of a menu item.

```
void SetControlMenuItem(ControlHandle ctl, short item,
   ConstStr255Param string)
{
   SetItem(GetControlMenu(ctl), item, string);
   if (GetControlValue(ctl) == item)
      ChangedControlMenu(ctl);
}
```

This routine is analogous to the SetItem routine of the Menu Manager. The ctl parameter is a handle to a popup menu control. The item parameter is the index of an item in the menu. The string parameter is the text to which to set the indexed item in the menu. First, the routine sets the text of the item to the string, using the routine GetControlMenu, which was described earlier, to access the control's menu handle. Then, if the currently selected item number in the popup menu control is the same as the item being altered, the routine calls ChangedControlMenu to force the popup menu control to update its display.

Disabling a Menu Item

A routine similar to SetControlMenuItem can be used to enable or disable an item in a popup menu control.

```
void EnableControlMenuItem(ControlHandle ctl, short item,   Boolean enable)
{
   if (enable)
      EnableItem(GetControlMenu(ctl), item);
   else
      DisableItem(GetControlMenu(ctl), item);
   if (GetControlValue(ctl) == item)
      ChangedControlMenu(ctl);
}
```

This routine enables the specified item in the menu used in a popup menu control.

Differences From the System CDEF

This section is intended for people who are already using the popup CDEF provided with System 7. While this popup CDEF is similar to the popup CDEF provided with System 7, the two CDEFs are not identical. Following are 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 menu control's triangle is drawn; the control'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 "Inside Macintosh: Toolbox Essentials".

Popup menus are colorized as specified by a control's color table or by a menu's color table resource. The popup menu control will be drawn in color if the control has a color table associated with it, or 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 is not 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 menu control is created, and the menu must be inserted into the menu list before the popup menu 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 menu is displayed. In the system CDEF, if several 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. Multiple selected items are not supported in the system CDEF or in my popup CDEF.

A popup menu control's maximum value is automatically set to the maximum number of items in the menu. If you change the number of items in the menu, there is no need to call SetControlMaximum. The control's maximum value will be updated the next time the CDEF is called.

Features in the System CDEF That are not Supported

The following features of the system CDEF are not supported by my CDEF:

popupTitleCenterJust	ignored

Other Differences

The system 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 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 CDEF, and can be used when you want to align several popup menus.

The appearance of controls drawn with my CDEF varies somewhat from the appearance of controls drawn with the system popup CDEF. The main differences in appearance follow.

- Color controls are supported in my CDEF.
- In the system CDEF, the triangle is disabled (drawn in gray) if the currently selected item is disabled, even though users can still click on the control to display other items in the menu. In my opinion, the triangle serves as one of several visual cues to users that they can click on the control and it will display a list of items. The triangle is therefore drawn in black (or a suitable color) even if the currently selected item is disabled. The triangle is drawn in gray (or a suitable color or pattern to indicate a disabled state) if the entire control is disabled.
- In the System CDEF, the control's title is drawn at the top of the control. In my CDEF, the control's title is centered vertically in the control. I don't see any compelling reason why the title should be drawn one way or the other, though I personally find the vertically centered title more aesthetically pleasing.
- The System CDEF draws the command-key equivalent of the currently selected item in the current selection rectangle. My CDEF does not draw the command-key equivalent.

Except for these differences, and the differences mentioned previously, the variations are minor, and probably won't be noticed by most users.

Bugs

Significant Bugs

This section lists significant bugs that must be fixed in the popup CDEF. The items are listed in approximate order of urgency.

(There are no known significant bugs.)

Moderate Bugs

This section lists bugs that are of moderate severity, and may therefore never be fixed because their existence is due to a "design decision" or they occur due to a useful "feature". The items are listed in approximate order of severity.

The code samples in this document have never been compiled.

There are too many part codes for adding color to a popup menu control.

Colorizing a popup menu control is too complicated.

Using popup menu controls is too complicated.

This CDEF suffers from *Featuritis Excessivitis*[2].

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.

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

Some basic configurations have not been tested, since I do not have access to the necessary hardware 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 was not able to implement or fix for one reason or another.

[2]The chronic accumulation of too many features.

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

Add a Version History file.

Add a picture to show the parts of a control.

Speed and memory usage could be improved.

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 erroneous 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 triangle 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. Stefan Kurth has also helped me refine the color display of the menu, and, specifically, helped fix a lack of visual feedback when a color popup menu control's title was hilited.

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.

Peter Berck (P.J.Berck@kub.nl) found a bug caused by a missing PenNormal, which resulted in the CDEF using the pen size settings of the control's port when drawing the triangle.

Contact Information

You can contact me at the following addresses (as of November 1994).

Postal mail:	Ari Halberstadt 9 Whittemore Road Newton, MA 02158-2105 USA
Electronic mail:	ari@world.std.com