CtlCTab structure

#include <Controls.h>

typedef struct **CtlCTab** {

		Size	Offset	Description
long	ccSeed;	4	0	Unused, value = 0
short	ccReserved;	2	4	Unused, value = 0
short	ctSize;	2	6	Elements in control's color table, minus 1
ColorSpec	ctTable[4];	4	8	Address of colorSpec records
} **CtlCTab**;		12		

typedef CtlCTab ***** **CCTabPtr**;
typedef CCTabPtr ***** **CCTabHandle**;

Notes: A colorSpec record holds an identifying short that says which part of the control it is coloring (cFrameColor, cBodyColor, cTextColor or cThumbColor) and three shorts that define the values of the color's red, green and blue components. The part identifiers can be listed in any order, but if a part is not identified at all, it will be drawn with the first color in the table.

If the color table is stored as resource type 'cctb' it still has this structure, and resides in the system in the same format as a 'clut' color table resource. A call to **InitWindows** loads the default color control resource 'cctb' = 0 into the application heap when your program starts up. The default is shared by all controls, regardless of type, and will be used in cases where there's no AuxCtlRec with an acOwner field pointing to the control being drawn. If you choose to set up your own color tables for your controls they can be as big as you want. You can also define them any way you want, except for using reserved part index numbers 1 through 127.

When a plain button is highlighted, it swaps colors 1 and 2, while highlighted check boxes and radio buttons change appearance but keep their colors the same. A deactivated button of any variety is indicated by dimmed text with no color change. The cThumbColor field is not used.

Scrollbars use the same partIdentifier and partRGB fields as the buttons, except that the cThumbColor field is filled in, while the cTextColor field is not. Highlighted arrows are an outlined foreground color while a deactivated scrollbar lacks an indicator and shows a solid background color.

Creating a new control with **NewControl** doesn't simultaneously create a new entry on the AuxList. Your controls will use the default color table unless you call **SetCtlColor** or create the control from a 'CNTL' resource by a call to **GetNewControl** and there is also an associated 'cctb' resource with the same ID in the resource file. If both types of resources exist, and they both have the same ID, **GetNewControl** will execute **SetCtlColor**.

Further, your control's colors will be limited to the eight accommodated

by the old-style grafPort unless your windows are opened in a cGrafPort by a call to **NewCWindow**. If you want controls with more than four colors you have to customize a 'CDEF' resource to recognize the extra colors and your applications 'CDEF' has to invoke **SysEnvirons** during the definition procedure to get the system configuration.

Finally, you dispose of the auxiliary control record by calling **DisposeControl**.