```
/* DLGTEMP.C -- version 1.00
*             This module supplies the necessary functions
*             used to create a dialog box template in global
*             memory and to pass the handle on to either the
*             DialogBoxIndirect or CreateDialogIndirect functions.
*/

#include <windows.h>
#include "dlgbox.h"

/*  forward references */
WORD   lstrlen( LPSTR );
void   SetStyleClass(int);

HANDLE   hDlgTemplate; /* Handle to current dialog template memory   */
WORD     wOffset;      /* Current memory offset (updated by CDH & CDI) */
BYTE     iItems;       /* number of items in dialog */
DLGITEM  DlgItem;      /* Dialog item structure */

/* ------------- Create Dialog Header ----------------------------*/

/*
*  This routine allocates a piece of global memory
*  and then fills in the dialog header structure and saves the
*  information in global memory.
*/

BOOL FAR PASCAL CreateDialogHeader( Style, ItemCount, X, Y, cX, cY, Resource, Class, Caption)
LONG  Style;     /* Dialog box Style */
BYTE  ItemCount; /* Control count for dialog box */
int   X;         /* Dialog box top left column */
int   Y;         /* Dialog box top row */
int   cX;        /* Dialog box width */
int   cY;        /* Dialog box height */
LPSTR Resource;  /* Dialog box resource string */
LPSTR Class;     /* Dialog box class string */
LPSTR Caption;   /* Dialog box caption */
{

  WORD   ResourceLength, /* Length of resource string */
         ClassLength,    /* Length of class string */
         CaptionLength;  /* Length of caption String */
  DWORD  dwMemLength;    /* Dialog header memory allocation length */
  LPSTR  lpHdrData;      /* Long pointer to locked dialog template memory */
  DLGHDR DlgHdr;         /* Dialog header structure */
  LPSTR  lpDlgHdr;       /* Long pointer to dialog header structure */
  int    i;              /* Loop index */

  /* Initialize memory offset */

  wOffset = 0;  /* set memory offset to ZERO */
  iItems  = 0;  /* set number of items in dialog to ZERO */

  /* Determine string lengths (including terminating null ) */

  ResourceLength = lstrlen( Resource ) + 1;
  ClassLength    = lstrlen( Class ) + 1;
  CaptionLength  = lstrlen( Caption ) + 1;

  /* Determine length of memory to allocate for dialog header */
  dwMemLength = (DWORD)( sizeof( DLGHDR ) + ResourceLength + ClassLength + CaptionLength );

  /* Allocate dialog template memory for dialog header and obtain handle */

  if ( !( hDlgTemplate = GlobalAlloc( GMEM_MOVEABLE | GMEM_ZEROINIT, dwMemLength ) ) ) {
     MessageBox( GetFocus(), (LPSTR)"GlobalAlloc Error", (LPSTR)"CDH", MB_OK );

     return FALSE; /* GlobalAlloc failed */
  }
```

```
 /* Lock allocated memory for modification */
 if ( lpHdrData = GlobalLock( hDlgTemplate ) ) {

    /* Set dialog header structure data to passed in parameters */
    DlgHdr.dtStyle     = Style;
    DlgHdr.dtItemCount = 1;      /* set to zero */
    DlgHdr.dtX         = X;
    DlgHdr.dtY         = Y;
    DlgHdr.dtCX        = cX;
    DlgHdr.dtCY        = cY;

    /* Get pointer to dialog header structure */
    lpDlgHdr = (LPSTR)&DlgHdr;

    /* Copy dialog header structure to allocated memory */
    for ( i = 0; i < sizeof( DLGHDR ); i++ )
       *lpHdrData++ = *lpDlgHdr++;

    /* Copy resource string to allocated memory */
    while ( ( *lpHdrData++ = *Resource++ ) );

    /* Copy class string to allocated memory */
    while ( ( *lpHdrData++ = *Class++ ) );

    /* Copy caption string to allocated memory */
    while ( ( *lpHdrData++ = *Caption++ ) );

    /* Adjust memory offset past memory allocated for dialog header */
    wOffset += (WORD)( sizeof( DLGHDR ) + ResourceLength + ClassLength + CaptionLength );

    /* Unlock allocated memory */
    GlobalUnlock( hDlgTemplate );

    return TRUE;  /* everything worked so far, return TRUE */
 }
 else {
    MessageBox( GetFocus(), (LPSTR)"GlobalLock Error", (LPSTR)"CreateDialogHeader", MB_OK );
    GlobalFree( hDlgTemplate ); /* free allocated memory */
    return FALSE;    /* return null handle indicating failure */
 }
}

/* ------------- Create Dialog Item ----------------------------*/
/*
*  This routine fills in the dialog item structure and
*  saves the information in global memory after resizing it.
*/

BOOL FAR PASCAL CreateDialogItem( iCtrlID, lStyle, Class, X, Y, cX, cY, Text, ExtraBytes )
int    iCtrlID;      /* Control ID               */
LONG   lStyle;       /* Control style            */
BYTE   Class;        /* Control class            */
int    X;           /* Control top left column   */
int    Y;           /* Control top row          */
int    cX;           /* Control width            */
int    cY;           /* Control height           */
LPSTR  Text;         /* Control text             */
BYTE   ExtraBytes;   /* Control extra bytes count */
{
 LPSTR    lpCtrlData;  /* Long pointer to locked dialog template memory  */
 LPSTR    lpDlgItem;   /* Long pointer to dialog control structure       */
 int      i;           /* Loop index                                     */
 WORD     TextLength;  /* Length of control text string                  */
 DWORD    dwMemLength;
 HANDLE   hCurDlgTemp;

 DlgItem.dtilX  = X;
 DlgItem.dtilY  = Y;
```

```
 DlgItem.dtilCX = cX;
 DlgItem.dtilCY = cY;
 DlgItem.dtilID = iCtrlID;

 if (Class == (BYTE)0)
   SetStyleClass( (int)lStyle);
 else {
     DlgItem.dtilControlClass = Class; /* default, may change in next function */
     DlgItem.dtilStyle = lStyle;
 }
 TextLength  = lstrlen( Text ) + 1;
 dwMemLength = (DWORD)(wOffset + sizeof(DLGITEM) + TextLength + sizeof(BYTE) );
 hCurDlgTemp = GlobalReAlloc( hDlgTemplate, dwMemLength, GMEM_MOVEABLE);

 if ( hCurDlgTemp == NULL) {
    MessageBox ( GetFocus(), (LPSTR)"global lock", (LPSTR)"Failed!", MB_OK);
    GlobalFree( hCurDlgTemp );
    return FALSE;
 }
 hDlgTemplate = hCurDlgTemp;

  /* Adjust pointer to reallocated memory bypassing existing data */

  if ( (lpCtrlData = GlobalLock(hDlgTemplate)) == NULL) {
    MessageBox ( GetFocus(), (LPSTR)"global lock", (LPSTR)"Failed!", MB_OK);
    GlobalFree( hDlgTemplate );
  }

  lpCtrlData += wOffset;

  /* Get pointer to dialog control structure */
     lpDlgItem = (LPSTR)&DlgItem;

  /* Copy dialog control structure to allocated memory */
     for ( i=0; i<sizeof( DLGITEM ); i++ )
        *lpCtrlData++ = *lpDlgItem++;

  /* Copy control test string to allocated memory */
     while ( ( *lpCtrlData++ = *Text++ ) );

  /* Copy extra byte count to allocated memory */
     *lpCtrlData = ExtraBytes;

  /* Adjust memory offset past memory reallocated for dialog control */
     wOffset += (WORD)( sizeof( DLGITEM ) + TextLength + sizeof(BYTE));

  /* Unlock reallocated memory */
     GlobalUnlock( hDlgTemplate );

     iItems++;       /* bump up number of items in dialog */

     return( TRUE );  /* return successful */
}

/* ------------- End Dialog Header ------------------------------ */

/*
*  This routine changes the number of items in the
*  dialog header and then returns a handle to the global memory.
*/

HANDLE FAR PASCAL EndDialogHeader()  /* end dialog header function */
{
 LPSTR    lpCtrlData;  /* Long pointer to locked dialog template memory  */

 lpCtrlData = GlobalLock( hDlgTemplate );
 if (lpCtrlData == NULL) {
   MessageBox ( GetFocus(), (LPSTR)"Global lock", (LPSTR)"Failed!", MB_OK);
   GlobalFree( hDlgTemplate );
```

```
 }

  /* 5th byte is address to # of items in dialog */
 lpCtrlData++;
 lpCtrlData++;
 lpCtrlData++;
 lpCtrlData++;

  /* set the number of items in control */
 *lpCtrlData = (BYTE)iItems;

 /* Unlock allocated memory */
 GlobalUnlock( hDlgTemplate );

 /* Return the handle to the Dialog Template */
 return (hDlgTemplate);
}


/* ------------- List of predefined dialog items ------------------- */

/*
*  This routine defines a list of predefined dialog items.
*  This allows the user to just pass a number to get the
*  attributes of a pre-defined control.
*  You can add to this list or modify this to best fit your needs.
*
*  Warning: If you change one of the styles, it could effect
*           other controls that expect the old behavior.
*/

void SetStyleClass(iStyle)
int  iStyle;
{
 /* if a iStyle is given then dtilControlClass will take on the default value. */

 switch ( iStyle ) {
    case DI0:     /* check box */
          DlgItem.dtilControlClass = BUTTONCLASS ;
          DlgItem.dtilStyle = BS_CHECKBOX | WS_TABSTOP | WS_CHILD | WS_VISIBLE ;
          break;
    case DI1:     /* icon */
          DlgItem.dtilControlClass = STATICCLASS ;
          DlgItem.dtilStyle = SS_ICON | WS_BORDER | WS_CHILD | WS_VISIBLE ;
          break;
    case DI2:     /* black box */
          DlgItem.dtilControlClass = STATICCLASS ;
          DlgItem.dtilStyle = SS_BLACKRECT | WS_CHILD | WS_VISIBLE ;
          break;
    case DI3:     /* rectangle */
          DlgItem.dtilControlClass = STATICCLASS ;
          DlgItem.dtilStyle = SS_BLACKFRAME | WS_CHILD | WS_VISIBLE ;
          break;
    case DI4:     /* left static text */
          DlgItem.dtilControlClass = STATICCLASS ;
          DlgItem.dtilStyle = SS_LEFT | WS_CHILD | WS_VISIBLE ;
          break;
    case DI5:     /* multiline edit box */
          DlgItem.dtilControlClass = EDITCLASS ;
          DlgItem.dtilStyle = ES_LEFT | ES_MULTILINE | ES_NOHIDESEL |
                        ES_AUTOVSCROLL | ES_AUTOHSCROLL |
                        WS_VSCROLL | WS_HSCROLL | WS_BORDER |
                        WS_TABSTOP | WS_CHILD | WS_VISIBLE ;
          break;
    case DI6:     /* list box - sorted */
          DlgItem.dtilControlClass = LISTBOXCLASS ;
          DlgItem.dtilStyle = LBS_STANDARD | WS_CHILD | WS_VISIBLE ;
          break;
    case DI7:     /* vertical scrollbar */
```

```
        DlgItem.dtilControlClass = SCROLLBARCLASS ;
        DlgItem.dtilStyle = SBS_VERT | WS_CHILD | WS_VISIBLE ;
        break;
    case DI8:     /* horizontal scrollbar */
        DlgItem.dtilControlClass = SCROLLBARCLASS ;
        DlgItem.dtilStyle = SBS_HORZ | WS_CHILD | WS_VISIBLE ;
        break;
    case DI9:     /* group box */
        DlgItem.dtilControlClass = BUTTONCLASS ;
        DlgItem.dtilStyle = BS_GROUPBOX | WS_TABSTOP | WS_CHILD | WS_VISIBLE ;
        break;
    case DI10:    /* Push button */
        DlgItem.dtilControlClass = BUTTONCLASS ;
        DlgItem.dtilStyle = BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD | WS_VISIBLE ;
        break;
    case DI11:    /* radio button */
        DlgItem.dtilControlClass = BUTTONCLASS ;
        DlgItem.dtilStyle = BS_RADIOBUTTON | WS_TABSTOP | WS_CHILD | WS_VISIBLE ;
        break;
    case DI12:    /* default push button */
        DlgItem.dtilControlClass = BUTTONCLASS ;
        DlgItem.dtilStyle = BS_DEFPUSHBUTTON | WS_TABSTOP | WS_CHILD | WS_VISIBLE ;
        break;
    case DI13:    /* left check box */
        DlgItem.dtilControlClass = BUTTONCLASS ;
        DlgItem.dtilStyle = BS_LEFTTEXT | BS_CHECKBOX | WS_TABSTOP | WS_CHILD | WS_VISIBLE ;
        break;
    case DI14:    /* 3 auto state button */
        DlgItem.dtilControlClass = BUTTONCLASS ;
        DlgItem.dtilStyle = BS_AUTO3STATE | WS_TABSTOP | WS_CHILD | WS_VISIBLE ;
        break;
    case DI15:    /* centered edit control */
        DlgItem.dtilControlClass = EDITCLASS ;
        DlgItem.dtilStyle = ES_CENTER | ES_MULTILINE | WS_BORDER | WS_TABSTOP |
                    WS_CHILD  | WS_VISIBLE ;
        break;
    case DI16:    /* right edit control */
        DlgItem.dtilControlClass = EDITCLASS ;
        DlgItem.dtilStyle = ES_RIGHT | ES_MULTILINE | WS_BORDER | WS_TABSTOP |
                    WS_CHILD | WS_VISIBLE ;
        break;
    case DI17:    /* left edit control */
        DlgItem.dtilControlClass = EDITCLASS ;
        DlgItem.dtilStyle = ES_LEFT  | WS_BORDER | WS_TABSTOP |
                    WS_CHILD | WS_VISIBLE ;
        break;

    case DI18:    /* listbox w/out sort */
        DlgItem.dtilControlClass = LISTBOXCLASS ;
        DlgItem.dtilStyle = LBS_NOTIFY | WS_BORDER | WS_VSCROLL |
                    WS_CHILD   | WS_VISIBLE ;
        break;
    case DI19:    /* center static text */
        DlgItem.dtilControlClass = STATICCLASS ;
        DlgItem.dtilStyle = SS_CENTER | WS_BORDER | WS_CHILD | WS_VISIBLE ;
        break;
    case DI20:    /* right static text */
        DlgItem.dtilControlClass = STATICCLASS ;
        DlgItem.dtilStyle = SS_RIGHT | WS_CHILD | WS_VISIBLE ;
        break;
    default:      /* left edit control */
        DlgItem.dtilControlClass = STATICCLASS ;
        DlgItem.dtilStyle = SS_LEFT | WS_CHILD | WS_VISIBLE ;
        break;
 } /* end switch */

}
```

```
WORD  lstrlen( lpszString )
LPSTR lpszString; /* String to check  */
{
  WORD Length; /* Length of string */

  for ( Length = 0; *lpszString++ != '\0'; Length++ );

  return( Length );
}
```