The Toolbar Library

Stephen Chung
stephenc@cunixf.cc.columbia.edu
Tim Liddelow
tim@kimba.catt.citri.edu.au

This set of C and C++ routines implement the slick "toolbar" found in Microsoft Word for Windows, Excel etc. They have been written and compiled using Borland C++ 2.0 and 3.0. They should however work fine with other compilers. Provided are C routines and a small C++ class.

Using the Toolbar library

Look at the sample program TBTEST.C for an example. That toolbar is part of a Japanese word processor. The icons have not been included because they are embedded in a large resource file with everything else. If you want some of them, send Stephen Chung email.

First of all, the toolbar is a 3-dimensional, horizontal bar containing **toolbar buttons**. You can add other kinds of child window controls onto the toolbar as well, such as combo boxes etc.

Before you do anything, you must define an array of TOOLBARICON. Each element in that array corresponds to one toolbar button. The fields should be set according to the following:

```
int id          /* The numeric ID of the button (0-255) */
int x, y        /* The X,Y position of the button, in pixels */
int width,
    height      /* The width/height of the button, in pixels */
int state       /* The initial state of the button:
                                  -1 = Disabled
                                   0 = Off
                                   1 = On
                                   2 = Grayed
                */
int cycle       /* The mode of that button, upon a mouse click:
                     0 = Leave the state alone
                           (Set it with a BM_SETSTATE message)
                   1 = Always undepressed, but flashed once
                   2 = Toggle On --> Off --> On
                   3 = Toggle On --> Off --> Gray --> On
```

```
                */

      char *disabled    /* The name for the DISABLED bitmap */
      char *undepressed /* The name for the OFF bitmap */
      char *depressed   /* The name for the ON bitmap */
      char *grayed      /* The name for the GRAYED bitmap */
      char *pressing    /* The name for the PRESSING bitmap */
```

You should leave the rest of the fields alone. They are used internally by the library routines.

Creating the Toolbar

In the C version, to create the toolbar, call CreateToolbar. This routine has the following parameters:

```
HWND CreateToolbar (HWND            parent,
                    int             x,
                    int             y,
                    int             width,
                    int             height,
                    int             thickness,
                    int             id,
                    int             nr_buttons,
                    HANDLE          hInstance,
                    TOOLBARICON     *icons,
                    char            *xcursor)
```

parent	Parent window handle
x, y	(x, y) position of toolbar in pixels relative to parent window
width, height	Width and height of toolbar in pixels
thickness	Thickness or fatness of the toolbar
id	The id of the toolbar
nr_buttons	The number of buttons on the toolbar
hInstance	Instance handle of the application
icons	Points to a **TOOLBARICON** array
xcursor	The name of a cursor for a disabled toolbar button

The above routine will return the handle for the toolbar. You can use this handle to add more child controls (see TBTEST.C).

If you are using C++, the prototype for the constructor is

```
void ToolBar::ToolBar (HWND parent,
                       int x,
                       int y,
                       int width,
```

```
                    int height,
                    int thickness,
                    int id,
                    int n,
                    HANDLE hInstance,
                   TOOLBARICON *icons,
                    char *xcursor)
```

To obtain the window handle using C++, simply use the member function Window().

Notes

The toolbar will send a **WM_COMMAND** message to its parent when one of its buttons is clicked. The following paremeters are passed:

message	WM_COMMAND
wParam	Low Byte = Toolbar ID High Byte = Toolbar button ID
lParam	Low Word = Toolbar button window handle High Word = BN_CLICKED

REMEMBER that even other child controls that you define (such as list boxes) will return with wParam = (Child ID << 8) | (Toolbar ID). This means that you are restricted to having 256 toolbars and 256 toolbar buttons and child controls on each toolbar. Well, life is tough, isn't it?

Support routines

These routines are listed in prototype format and their parameters are explained. Note that the C++ equivalent class member functions do not always required all the parameters of the corresponding C one.

You enable and disable toolbar buttons by calling **EnableToolbarButton** with the following parameters:

```
void ToolBar::EnableToolbarButton (int    child,
                                   BOOL   Enabled)

void EnableToolbarButton (HWND      hwnd,
                          int       child,
                          BOOL      Enabled)
```

hwnd	Toolbar window handle
child	Id of the toolbar button
Enabled	Nonzero enables button, zero disables

You can also set or query the state of a toolbar button by sending the TOOLBAR BUTTON a **BM_GETSTATE** or **BM_SETSTATE** message, just as you would for a normal push button. You can also send the BM_GETSTATE and BM_SETSTATE messages to the toolbar's window procedure, with the toolbar button's ID passed in lParam.

If, for some twisted reason, you want to modify the characteristics of the toolbar button dynamically, you can first call GetToolbarButton to fill in a TOOLBARICON structure:

```
HWND ToolBar::GetToolbarButton (int             child,
                                TOOLBARICON     *icon)

HWND GetToolbarButton (HWND         hwnd,
                       int          child,
                       TOOLBARICON  *icon)
```

hwnd	Toolbar window handle
child	Id of the toolbar button
icon	Pointer to a **TOOLBARICON** structure to store the button information

This routine will return the window handle of the toolbar button. Now you can change the settings and then call **ModifyToolbarButton**:

```
void ToolBar::ModifyToolbarButton (HWND         hwndButton,
                                   TOOLBARICON  *icon)

void ModifyToolbarButton (HWND            hwndButton,
                          TOOLBARICON     *icon)
```

hwndButton	Toolbar **button** window handle
icon	The TOOLBARICON structore containing the new settings

There is also a neat little routine called Create3DEffect which will make any rectangle within any window look like a 3-dimensional bar:

```
void Create3DEffect (HDC      hdc,
                     RECT     *rect,
                     int      thickness,
                     int      style)
```

hdc	Handle to some device context to draw on
rect	Pointer to a rectangle defining the area to make 3D. If *rect* is NULL, the entired window is used.

C++ Example

```
foo()
{
      TOOLBARICON Icons[] = { ..... };
      TOOLBARICON IconInfo;

      ToolBar t(hParent, 10, 10, TOOLBAR_WIDTH, TOOLBAR_HEIGHT,
                        TOOLBAR_THICKNESS,      TOOLBAR_1,      NICONS,
hInstance,                    Icons, NULL);
      ...
      ...
      t.Hide();
      ...
      ...
      t.EnableToolbarButton(2, FALSE);
      ToolbarIconWindowHandle = t.GetToolbarButton(2, &IconInfo);
      ToolbarWindowHandle = t.Window();
      ...
      ...
}
```

If you want to change the name of the toolbar window classes, they are defined in **toolbar.h**.

Remember to export ToolbarProc and ToolbarButtonProc in your .def file of course.

Afterwords

Any questions and/or bug fixes, please send email to:

Stephen Chung stephenc@cunixf.cc.columbia.edu
or Tim Liddelow tim@kimba.catt.citri.edu.au

If it bounces, then try schung@cogsci.Berkeley.EDU

Have fun!