Tool Tips Toolbar Help LIbraray

****
*** PLEASE READ CAREFULLY ***
****

There are almost no restrictions for using this source code. You may use this code in any of your applications or pass it along to anyone else that may need this functionality. The only rule for using this code or any derivative of it, is that it must remain FREEWARE. All of us who have worked on this project think that CompuServe is a great place to exchange ideas and learn how to do things that we would otherwise have trouble figuring out. Everyone should give back; at least a little. Therefore we are giving this. Please do not take credit and/or remuneration for what has been provided as FREEWARE. You can, of course, use this code in any commercial or personal application. Since the code is FREEWARE, you may even include it in 'source code sales' as long as this source code does not represent more than ten percent (10%) of the entire project.

The authors retain ownership of this code based on the restrictions listed herein. By using this source code you agree to these restrictions.

Basically, we're giving this away and because of that EVERYONE should have a chance to receive it as FREEWARE.

Adding Toolbar Tips to Your Applications

Tool Tips are those little yellow boxes that popup automatically when your mouse is positioned over a toolbar button or control. If you have used any of the current Microsoft, Lotus, WordPerfect, or other top end products, you will recognize this as a very useful feature. Adding this functionality to your programs gives them that 'professional' feel. That is why we developed this special class (CToolTipBar) as a replacement for the CToolBar class included with MFC. This class was originally written by Ken Walters. Actually, Ken says that he first developed this class in response to a Borland ad that positioned OWL over MFC by virtue of fly by tool tips. Well, this class takes that edge away, and then some. Since its inception, Ken Walters, Tom Serface and Bob Cozzi have worked together to add features and functionality to the code. Tool Tips Version 2.0 is the result of that effort. We've tried to make this as easy to implement as possible without sacrificing advanced functionality. Tool Tips works with all kinds of toolbars, status bars, and palette bars, positioned anywhere on the screen. Tool Tips automatically works with different sized mouse cursors, windows, and ensures that tips are not displayed when the toolbar window does not have complete focus. There are a lot of tricky programming techniques used here. You may want to peruse the code, if not to learn about Tool Tips, perhaps to learn a little more about MFC.

To illustrate the use of Tool Tips, we've set up a sample application called TOOLTEST. This document uses TOOLTEST as an example of using the Tool Tips functions. You don't need to implement all of the functionality to make these routines useful. Therefore, this document describes TOOLTEST in various stages.

Note: In this example we continually refer to copying some stuff from another application. That is because we had already implemented these features in other applications and, to save time, we didn't want to do it over. We plan to use these steps when we set up new applications in the future so this document is serving a double purpose. In this case your 'other application' is TOOLTEST. App Studio is great for grabbing dialogs, strings, bitmaps, etc. from other application's resource files. For example, to copy an entire dialog you just highlight the name (you don't have to display the dialog), and select Edit/Copy then switch to another opened .RC file's window (using Ctrl+Tab) and paste it in using Edit/Paste. You can even copy more than one dialog by marking several (using Shift+Left Mouse Button to mark extra ones). The same works for string tables, bitmaps, menus, and other resources.

This is kind of overkill for an example, but here's the steps we went through to create the TOOLTEST program. To create the TOOLTEST application we:

1. Created an MDI application using AppWizard. (An SDI would have worked as well.)

2. Copied the following files into the TOOLTEST directory:

    Required:

    TOOLTIP.H
    TOOLTIP.CPP
    POPUPTIP.H
    POPUPTIP.CPP

3. Added:

    #include "tooltip.h";

  to the top of MAINFRM.H

4. Changed CToolBar to CToolTipBar in MAINFRM.H

Note: You can use CToolTipBar in any classes that you would have normally derived from CToolBar. That way you can get Tool Tips on all of your tool bars, palette bars, and style bars.

5. Added files TOOLTIP.CPP and POPUPTIP.CPP to the application's project file using the Add button in Project/Edit in the PWB.

6. Recompiled.

That's really all you have to do to get the default implementation of Tool Tips. You can make this into a library if you would like, but since it really is not that much code (especially in numbers of files) it is just as easy and a lot more convenient to include it in your the project. That way, if you compile for debugging, the Tool Tips files get recompiled as well. Having the source code makes it possible to include this in 16 or 32 bit applications using any memory model. We hope we didn't leave anything out. It compiles and all of the source code is here. So ...

How Tool Tips Works

Tool Tips uses the following logic:

When it is determined that the mouse is over a toolbar button:

1. Tool Tips tries to get the menu string which corresponds to the toolbar button. Most of the time there is a menu item for a toolbar button so this works pretty well.

2. Tool Tips strips off the '&' (underline), the '...' (ellipse), and the /tKey (accellerator key) from the menu string. This makes the string look better in the tip box and is purely cosmetic.

3. Displays the tip.

4. If there is no menu item for the toolbar button Tool Tips looks for a resource string with the same uId as the button. In TOOLTEST we added IDD_SAMPLEBUTTON and string resource IDD_SAMPLEBUTTON to illustrate this. If a string is found it is scanned for text in ()s. That is assumed to be the tip subset. If no text is found in ()s then the first word on the line is used. This is also useful for other types of controls on a toolbar (e.g., COMBOXBOX).

5. If no menu item or string resource is found for the toolbar the words "Not Defined" are displayed in the tip box. When you see this message (hopefully during your test cycle) you know that you need to add in a string table or menu entry for that button. When the mouse moves off the toolbar, or the users clicks a mouse button, the tip window is destroyed and the screen under it is restored. The Tool Tips engine works by checking in OnIdle(). Tool Tips continually pumps itself messages so that it can continue call OnIdle().

Tool Tips Public Member Functions

We added these CToolTipBar member functions so that you can use them in your programs to control the Tool Tip functions.

// Gets the current wait time in milliseconds. By default Tool Tips uses
// 500 milliseconds. This value can be changed with the SetWait() member
// function.
UINT CToolTipBar::GetTipWait ();

Example: UINT SaveWait = GetTipWait();

// Sets the amount of time (in milliseconds) to wait after a button is
// identified before popping up the tip window.
// This option is saved in the app.ini file and automatically read when the
// program is started at a later time.
void CToolTipBar::SetTipWait (UINT nNewWait);

Example: SetTipWait(500);

// Returns the current setting for status line messages
BOOL CToolTipBar::GetFlyby();

Example: BOOL SaveFlyby = GetFlyby();

// Sets whether or not to display popup tip messages (in the yellow box).
void CToolTipBar::SetFlyby (BOOL bFlyby);

Example: SetFlyby(TRUE);

// Returns the current setting for popup tip messages
BOOL CToolTipBar::GetTips();

Example: BOOL SaveTips = GetTips();

// Sets whether or not to display popup tip messages (in the yellow box).
void CToolTipBar::SetTips (BOOL bTips);

Example: SetTips(TRUE);

// Returns the current box style.
// TTIPS_SQUARESTYLE (0) Square box with black on yellow text
// TTIPS_ROUNDSTYLE (1) Rounded box with black on yellow text
// TTIPS_3DSTYLE (2) 3D frame with black on gray text
UINT CToolTipBar::GetTipStyle();

Example: UINT SaveTipStyle = GetTipStyle();

// Sets the current box style.
// TTIPS_SQUARESTYLE (0) Square box with black on yellow text
// TTIPS_ROUNDSTYLE (1) Rounded box with black on yellow text
// TTIPS_3DSTYLE (2) 3D frame with black on gray text
void CToolTipBar::SetTipStyle (UINT bNewTipStyle);

Example: SetTipStyle(TTIPS_ROUNDSTYLE);

// Gets the current setting of the use systems font flag. TRUE means use the
// normal font (like in the systems menu) FALSE means use the smaller one which
// most applications are using
BOOL CToolTipBar::GetTipFont();

Example: BOOL SaveTipFont = GetTipFont();

// Sets the current systems font flag. TRUE means use the
// normal font (like in the systems menu) FALSE means use the smaller one which
// most applications are using
void CToolTipBar::SetTipFont (BOOL bNewTipFont);

Example: SetTipFont(FALSE);

// Get the current toolbar display status FALSE = off TRUE = on. This
// function returns whether or not the toolbar is currently being displayed.
BOOL CToolTipBar::GetToolbarDisplay();

Example: SaveDisplay = GetToolbarDisplay();

// Sets the display flag FALSE = off TRUE = on. This routine actually does
// the showing or hiding of the toolbar window and sets and internal
// variable that is saved in the app.ini file with the rest of the settings.
void CToolTipBar::SetToolbarDisplay(BOOL bNewToolbarDisplay);

Example: SetToolbarDisplay(TRUE);

// Retrieves the toolbar options from the app.ini file. This is done
// automatically in the toolbar constructor if the file exists. The
// function is provide in case you want to re-read the settings.
void CToolTipBar::GetToolbarOptions();

Example: GetToolbarOptions();

// Writes the current settings to the app.ini file. This is done
// automatically in the destructor. The function is provide so that you
// can save the settings at will.
void CToolTipBar::WriteToolbarOptions();

Example: WriteToolbarOptions(SaveFlyby, SaveTips, SaveBoxStyle);

// This function is for adding in a control that is not native to the
// toolbar, but is positioned in the toolbar. For example, on the VC++
// toolbar there is a find COMBOBOX. You call this function with the
// index of the separator you used (the one that the control overwrites.)
// There is a good example of using controls in a toolbars in directory
// \MSVC\MFC\CTRLBARS. Remember that the value of nIDXControl is the
// index, in button order, of the ID_SEPARATOR. For example, if the
// separator is the second control on the line you would enter 2 as the
// argument to this function.
void CToolTipBar::AddCustomButton(UINT nIDXControl);

Example:

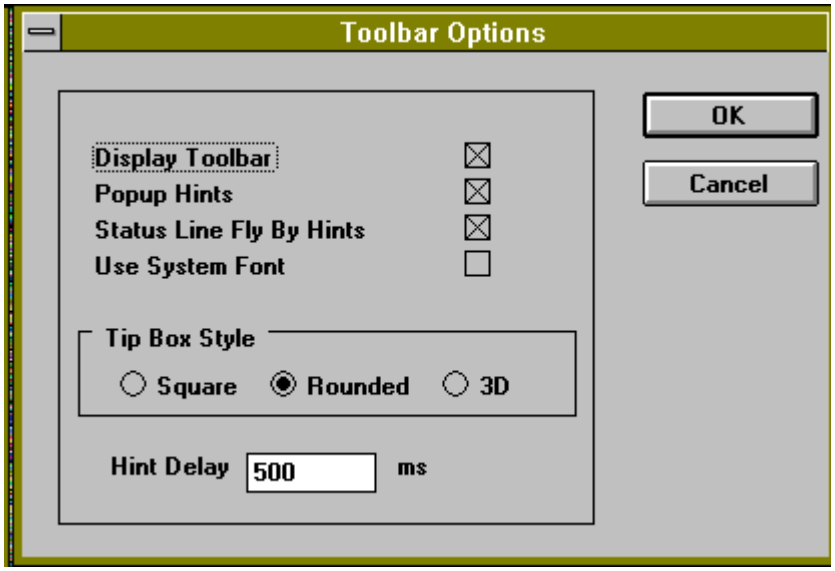
```
//
// This code is from the CTRLBARS example
//
static UINT BASED_CODE styles[] =
{
            // same order as in the bitmap 'styles.bmp'
            ID_SEPARATOR, // Space before the control
            ID_SEPARATOR, // This is where the control goes, it is index 1
            ID_SEPARATOR, // Space after the control
            ID_STYLE_LEFT,
            ID_STYLE_CENTERED,
            ID_STYLE_RIGHT,
            ID_STYLE_JUSTIFIED,
};

// Create the combo box and changes the width of the separator to 100
// so the combo box will fit into the toolbar
m_wndStyleBar.SetButtonInfo(1, IDW_COMBO, TBBS_SEPARATOR, 100 );
// Design guide advises 12 pixel gap between combos and buttons so
// change the width of the separator that comes after the combo box.
m_wndStyleBar.SetButtonInfo(2, ID_SEPARATOR, TBBS_SEPARATOR, 12 );

//
// Now create the combo box in the toolbar.  Note: m_comboBox is a
// member of Class CStyleBar which is derived from CToolTipBar
CRect rect;
m_wndStyleBar.GetItemRect( 1, &rect);
rect.top = 1;
rect.bottom = rect.top + 100;
if (!m_wndStyleBar.m_comboBox.Create(
             CBS_DROPDOWNLIST|WS_VISIBLE|WS_TABSTOP,
              rect, &m_wndStyleBar, IDW_COMBO))
{
TRACE("Failed to create combo-box\n");
        return FALSE;
}
//
// Added this line
// Note: Remember to create a string table entry with the same ID
//       as the control (for the tip).  In this case it is IDW_COMBO.
//
m_wndStyleBar.AddCustomButton(1); // COMBOBOX index
```

Adding a Tool Tips Options Dialog

To show how to use these functions, we added the following dialog screen:



You will probably want to customize the dialog a bit for your application, but this is a pretty good head start. This dialog gives the user the option of setting Tool Tip parameters went through the following steps (*note: this dialog was created previously and functions in another application so we just copied them here.*) You will be copying from TOOLTEST to your application.

1. Copy the tool options dialog files to your application.

   For Tool Tips Options Dialog:

   TOOLOPTI.H
   TOOLOPTI.CPP

2. Copy the .RC files stuff by opening TOOLTEST.RC and your applications' .RC file and using Edit/Copy and Edit/Paste on the Dialog. The dialog uId is IDD_TOOLBAR.

3. In your application, change the default menu View/Toolbar to Toolbar Options... and change the uId to IDD_TOOLBAR. We changed both menus since this is an MDI. Make sure you copy the dialog before adding its uId to the menus.

4. Create an OnToolbar() function in the CMainFrame class using ClassWizard for IDD_TOOLBAR which effectively overwrites the framework implementation, but it isn't needed anymore. The Tool Options dialog allows you to turn on/off the toolbar which is all the framework did anyway.

5. Edit the code for OnToolbar() in MAINFRM.CPP and added a call to:

   SetToolbarOptions(TRUE); // display dialog

This is a function that we wrote to control the Tool Options dialog. You will copy it in a later step. The TRUE option means that you want the function to actually display the dialog. FALSE means just set the current toolbar options. We added the FALSE option so that we can call this function to set up the initial state of the toolbar in case the user previously leaves the program with the

toolbar switched off. You can't show or hide the toolbar in the constructor because at that point it is always hidden.

6. Added:

    #include "toolopti.h"

to the top of MAINFRM.CPP. You need this so that SetToolbarOptions can create a CToolbarOptions dialog object.

7. Copied the SetToolbarOptions() routine from another app's MAINFRM.CPP and the prototype for SetToolbarOptions() from its MAINFRM.H. You can get these by opening both files and just marking and copying the lines.

8. Put the call:

    SetToolbarOptions(FALSE); // don't display dialog

right after the toolbar is created in MAINFRM.CPP OnCreate(). This is the call that hides the toolbar if the user left it off last time. The CToolTipBar constructor reads the .ini file to find out the toolbar state. You can look in TOOLTEST's MAINFRM.CPP if you are confused about where to put this call. If is pretty much the same in an MDI or SDI application.

9. Change:

    #include "tooltest.h" to #include "yourapp.h"

  in TOOLOPTI.CPP

10. Recompile.

At this point you have an MDI app with Tool Tips.

Note: We took the <-? prompt off the toolbar because it doesn't have a default menu implementation. It would be as easy to add an entry for Context Help in the help menu. We also added in a dummy 'B' button to test the button without menu item tip. We added in IDD_SAMPLEBUTTON to the symbols using Edit/Symbols in the App Studio menu, and to the string table, also using App Studio. Then we manually added IDD_SAMPLEBUTTON to the toolbar list in MAINFRM.CPP. We manually created a do nothing function called OnSampleButton() in MAINFRM.CPP and added a prototype for it in MAINFRM.H. We did this manually by adding and entry called IDD_SAMPLEBUTTON into the string table using App Studio.

****
****

If all you care about is TOOL TIPS STOP Here!!! The rest of this file is information about Splash Screens and About Boxes.

Adding a Custom About Box and Intro Splash Screen

The rest of this document describes how to do a splash screen and MS style About Box. It makes applications look fancier and happened to be in TOOLTEST so we thought we'd describe the process. We are only describing it here so that anyone who is interested will, hopefully, be less confused about how it works.

A lot of this code comes from the SuperPad example. There are some changes and it is put into files so you can pack it around from program to program. It takes about 2 or 3 minutes to add this functionality to a project.

1. Copy the splash screen/about box files:

    SPLASH.H
    SPLASH.CPP

2. Copy the About Box and Splash dialogs from another application using App Studio Edit/Copy and Edit/Paste. You can use the TOOLTEST application's .RC file to copy from.

3. Copied the IDS_ strings that are used in the about box dialog from the other app to the new application (in this case TOOLTEST).

  IDS_DISK_SPACE_UNAVAIL
  IDS_DISK_SPACE
  IDS_MATH_COPR_NOTPRESENT
  IDS_AVAIL_MEM
  IDS_RESOURCE_FREE

You can mark all of these and copy them from TOOLTEST.RC to your application using the Edit/Copy and Edit/Paste functions in App Studio.

4. Changed the dialog caption and titles and stuff in IDD_ABOUTBOX and IDD_SPLASH using App Studio. Test are just dialogs, so you can rearrange them anyway you want. Just don't get rid of the big button or the items in () like (Avail Mem).

5. Changed:

    #include "tooltest.h" to #include "yourapp.h"

in SPLASH.CPP

6. Remove everything that has to do with the about box, like the definition for class CAboutDlg, the constructor, destructor, etc., except the call to CTooltest::OnAppAbout(), from TOOLTEST.CPP. This call will be CYourapp::OnAppAbout() in your application. It is the function that the menu item and toolbar button are hooked into. The CAboutDlg is now in SPLASH.H and SPLASH.CPP.

7. Add:

    #include "splash.h"

to TOOLTEST.H

8. Add lines marked:

```
// Added these lines to do the splash screen stuff
// ********************************
  ...
// ********************************
```

to TOOLTEST.CPP and TOOLTEST.H. If you look in TOOLTEST.H and TOOLTEST.CPP you will see lines that are marked this way. You'll want to copy all of them from the TOOLTEST application and put them in the same place in your files. There are several occurrences of these types of lines so scan the entire file.

9. In App Studio we selected a better icon than the AFX given to you by default. You can, of course, use whatever icon you want. The routine automatically uses the program's main icon, resizes it, and adds the shadow effect. It puts the icon whereever the big button is in the IDD_ABOUTBOX and IDD_SPLASH dialogs.

9. Add file SPLASH.CPP to the application using the Add button in Project/Edit in the PWB.

10. Recompile.

****
****

Hope this helps you out some,

The Authors ...