

# The Advanced Toolbar Control (ATC) from Alien Technologies

*A breakthrough library that adds advanced toolbar features to an application in less than a day.*



Version 1.5

ATC is designed to assist programmers in supporting the advanced toolbar features appearing in today's best-selling Windows applications. The ATC programming interface is extremely easy to use and designed to easily integrate into an application framework in less than a day.

The 16-bit ATC package consists of Dynamic Link Libraries (DLL's), libraries, and header files necessary to create Windows 3.1 applications using Microsoft and Borland compilers.

The 16/32-bit ATC package adds DLL's necessary to create Windows NT and Win32s applications.

The 16/32-bit Source ATC package adds the complete source code to the ATC libraries.

[New Features!](#)

[Using ATC](#)

[User Features](#)

[Programmer Features](#)

[Ordering Information](#)

[Licensing Agreement](#)

[About Alien Technologies](#)

© 1994 Alien Technologies. All rights reserved.

Advanced Toolbar Control is a trademark of Alien Technologies.

Microsoft, MS, and MS-DOS are registered trademarks and Visual C++, Windows, Windows NT, Win32s, and Win32 are trademarks of Microsoft Corporation.

Borland is a registered trademark of Borland International, Inc.

## Using ATC

The included 16-bit ATC libraries allow you to build 16-bit Windows 3.1 applications. If the 32-bit libraries were purchased then you can also create 32-bit Windows NT or Win32s (running under Windows 3.1) applications.

### 16-bit applications

The 16-bit ATC libraries are compatible with Borland and Microsoft compilers for creating Windows 3.1 applications. The ATC library is mixed-model compatible which means your application can be compiled using the small, medium, or large memory model.

The following is the list of files that are involved with a 16-bit ATC application:

atc.h	This header file should be included by all program source code modules that call ATC functions. This is compatible with 16- and 32-bit applications. This file can be found in the installed ATC\INCLUDE directory.
atc16.dll	Run-time library for 16-bit applications. This file must be available to the application by placing the file in the same directory as the executable or in the Windows directory. This file can be found in the installed ATC\REDIST directory.
atc16.lib	Import library for use with Microsoft or Borland compilers when used to create 16-bit applications. Link this in with your project. This file can be found in the installed ATC\LIB directory.

### 32-bit applications

The 32-bit ATC libraries are compatible with Borland and Microsoft compilers for creating Windows NT or Win32s applications.

The following is the list of files that are involved with a 32-bit ATC application:

atc.h	This header file should be included by all program source code modules that call ATC functions. This is compatible with 16- and 32-bit applications. This file can be found in the installed ATC\INCLUDE directory.
atc32.dll	Run-time library for 32-bit applications. This file must be available to the application by placing the file in the same directory as the executable or in the Windows (for Win32s) or Windows NT directory. This file can be found in the installed ATC\REDIST directory.
atc32ms.lib	Import library for use with Microsoft Visual C++ for Windows NT when used to create 32-bit applications. Link this in with your project. This file can be found in the installed ATC\LIB directory.
atc32bc.lib	Import library for use with Borland C++ 4.0 when used to create 32-bit applications. Link this in with your project. This file can be found in the installed ATC\LIB directory.

Note that the .LIB files are **not** compatible between Borland and Microsoft compilers. However, the .DLL files can be used with an application generated with either compiler.

## **New Features**

*ATC continues to deliver high-quality, innovative features for users and developers!*

The following features are new to version 1.5:

- Independent toolbar systems can exist on multiple windows with their own private buttons, toolbars, and dock.
- Text can be automatically drawn on the toolbar buttons providing additional help for the user. The user can control the display of button text at run-time and the buttons, toolbar, and dock will automatically shrink or expand to accommodate the text.
- Buttons can be classified into categories for the Insert Item dialog box. The user can choose which category of buttons should be displayed to assist in tracking down a particular button.
- Button images are no longer forced to exist along one row in the bitmap resource. The bitmap can contain multiple rows of button images allowing an almost unlimited number of buttons!
- ATC now correctly works as a server for multiple simultaneous applications or multiple instances of the same application.
- An MFC wrapper class has been created for ATC. Now ATC and MFC can work together in the same application!

## User Features

*ATC supports the latest and most advanced toolbar features appearing in today's best-selling Windows applications.*

The following features are available to the application user:

Toolbar Dock

Popup Toolbar Menu

User Configurable

Quick Help

Button Text

Floating Toolbar Windows

## Toolbar Dock

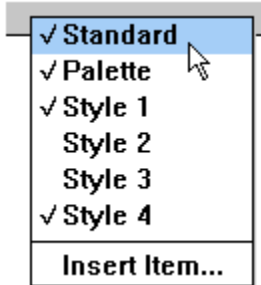
The dock's purpose is to contain multiple toolbars in a static window at the top of the screen. The toolbars within the dock can be rearranged by the user by clicking and dragging the toolbars around with the left mouse button.

The dock will automatically expand down the window as more toolbars are added. The toolbars are wrapped based on the maximum width of the application window.

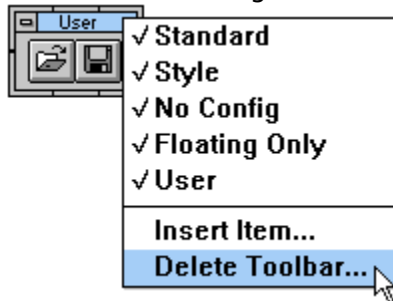


## Popup Toolbar Menu

Pressing and releasing the right mouse button on the toolbar or dock will cause a toolbar popup menu to appear. This menu allows the user to make a toolbar visible or invisible or insert items into a toolbar.



When the mouse is clicked on a user-created toolbar a new item appears on the bottom of the menu allowing the user to delete the toolbar.



## User Configurable

An ATC toolbar is highly configurable by the user. New toolbar items and separators can be inserted and removed from the toolbar. Existing items can be completely rearranged on the same toolbar or between multiple toolbars. User-created toolbars can also exist allowing the user to create and delete entire toolbars to suit their needs.

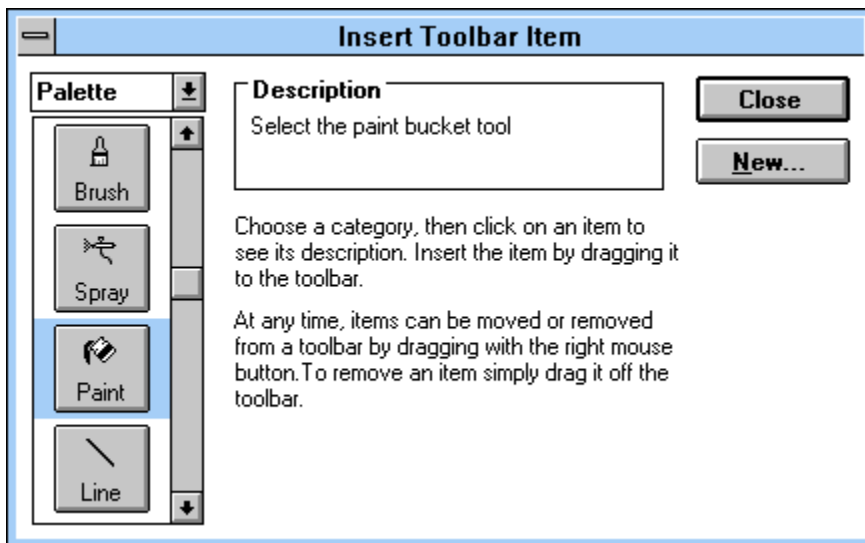
The toolbar buttons can be rearranged by the user during run-time via a drag and drop interface. The drag and drop interface is active at *all* times. Simply use the right mouse button to click and drag a toolbar item to move it to its new location.

Dragging an item just a small amount will insert or remove separators between items.

Dragging an item off the toolbar will remove the item.

Dragging an item off a toolbar and onto another toolbar will remove it from the first toolbar and insert it into the other. You can perform a copy instead of a move by hold the Control key down when you first click on an item to begin the drag process.

To insert an item click the right mouse button down and release it anywhere on the toolbar or dock to bring up the popup toolbar menu. Choose the Insert Item menu item to bring up the following dialog box.



From here you can click on an item from the listbox and see information about that item displayed in the assigned box. Items can be broken up into categories allowing the user the quickly find a particular button. The category can be changed in the combo box located above the listbox. Dragging an item with the left or right mouse button and releasing it on a toolbar will insert the item.

Clicking on the New button will allow you to create a new, user toolbar.

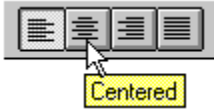


After entering a toolbar title and pressing return, the empty new toolbar will appear allowing you to easily drag and drop items to that window. The new window can be deleted by choosing the Delete Toolbar menu item that appears on the popup toolbar menu when you click with the right mouse button on that toolbar.



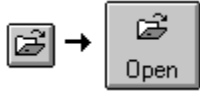
## Quick Help

Quick Help provides an immediate description of a toolbar item for the user. Resting the mouse cursor over a toolbar item for a short amount of time will cause a small help window to appear. This window disappears as soon as the mouse is moved or the buttons are pressed.



## Button Text

Button text can be drawn directly on the toolbar buttons to further aid the user in determining the purpose of a button. One or two words can easily fit under the button image and the button outline itself will expand or contract. The user can control the display of button text at run-time and the buttons sizes will change automatically.



## Floating Toolbar Windows

Using the left mouse button, drag the toolbar off its dock and move it elsewhere on the screen. The toolbar will turn into a resizable floating window and the dock window will readjust its shape.



To return the toolbar back to the dock window simply drag it to the dock and release. The docked version of the toolbar will be inserted at the drop location on the dock.

The user can also double-click on a toolbar to have it immediate jump between its docked and floating state. Clicking once on the standard close box in the top/left corner of the floating window will hide the window. Use the popup toolbar menu to make the window visible again.

# Programmer Features

*A quick tour of the API provided by ATC for the application programmer.*

The following topics describe how easy and powerful ATC is for the application developer.

[Initializing](#)

[Global Settings](#)

[Installing Systems](#)

[Defining Buttons](#)

[Defining Categories](#)

[Defining Docks](#)

[Defining Toolbars](#)

[Button Styles](#)

[Button States](#)

[Toolbar Settings](#)

[Status Bar Messages](#)

[Application Sizing](#)

[Layout Saving](#)

[Layout Restoration](#)

["Toolbars" Menu](#)

[MFC Support](#)

[Object-Oriented Design](#)

[Robust Source](#)

## Initializing

An instance of the ATC DLL is initialized and terminated with the `ATC_Initialize` and `ATC_Terminate` functions. An instance should be started by multiple instances of an the same application or by different applications.

When initializing ATC you need to define the instance handle that ATC should use to retrieve bitmaps and a function callback that ATC requires to obtain strings.

A commercial quality application may separate resources common to all foreign languages from those specific to a particular language. For example, the button bitmap may reside in a common resources DLL but strings may reside in a special DLL which contains all language-dependent strings.

The callback function is needed to supply ATC with the following strings:

- If user-configurability will be supported then description strings based on an item's command ID must be supplied. These are used in the Insert Item dialog box.
- If Quick Help will be supported then Quick Help strings based on an item's command ID must be supplied.
- If Button Text will be supported then Button Text strings based on an item's command ID must be supplied.
- If item categories will be supported then strings for category headings must be supplied.

See the demo application source code installed with ATC and the Sample Code that follows for more information.

## Sample Code

```
static ATCLOADSTRING glpfnATCLoadString; // The ATC load string function

// Make the callback function exportable with MakeProcInstance
glpfnATCLoadString = (ATCLOADSTRING)MakeProcInstance (
    (FARPROC)ATCLoadStringProc, ghInstance);

// Initialize ATC
ATC_Initialize (ghInstance, (ATCLOADSTRING)glpfnATCLoadString);

// This is the load string function for ATC when it needs to get a
// quick help or description string.
int ATC_FUNCTION // Length of the string
ATCLoadStringProc(
    UINT uiCmdID, // The item command ID
    ATC_WANT_STRING eWantString, // The type of string ATC wants
    char FAR *szBuffer, // The given buffer to fill
    int iSize) // The size of the given buffer
{
    // ATC wants a quick help string
```

```
if (eWantString == ATC_HELP_STRING)
    return (LoadString (ghInstance, OFFSET_HELP_IDS + uiCmdID,
                        szBuffer, iSize));

// ATC wants a description string for the Insert Item dialog
else if (eWantString == ATC_DESC_STRING)
    return (LoadString (ghInstance, uiCmdID, szBuffer, iSize));

// ATC wants a text string to draw on the button
else if (eWantString == ATC_TEXT_STRING)
    return (LoadString (ghInstance, OFFSET_TEXT_IDS + uiCmdID,
                        szBuffer, iSize));

// ATC wants a string to display the different item categories
else if (eWantString == ATC_CATEGORY_STRING)
    return (LoadString (ghInstance, uiCmdID, szBuffer, iSize));

// If I don't handle the string type then just return 0
else
    return (0);
}
```

## Global Settings

The ATC system can be configured with the following global, system-independent options:

- Drag and drop configuration of toolbar items can be enabled or disabled.
- The display of Quick Help can be enabled or disabled.
- The display of Button Text can be enabled or disabled.

### Sample Code

```
// Turn off quick help
ATC_ShowQuickHelp (FALSE);

// Turn off drag and drop configuration of toolbar buttons
ATC_SetUserConfig (FALSE);

// Turn off button text
ATC_ShowButtonText (FALSE);

// Is quick help on?
if (ATC_IsQuickHelpOn ()) ...

// Is the user able to configure the toolbar items?
if (ATC_UserConfigOn ()) ...

// Is button text on?
if (ATC_IsButtonTextOn ()) ...
```

## Installing Systems

A system should be installed and removed with the `ATC_InstallSystem` and `ATC_RemoveSystem` functions. Systems are used to install buttons, toolbars, and a dock for a particular window.

Once ATC is initialized toolbar systems can be installed for a given application window. This window will receive the `WM_SIZE` and `WM_COMMAND` messages sent by ATC.

Systems were created so that multiple application windows can independently control their own private buttons and toolbars. Each window will have its own dock which accepts only toolbar and buttons designed for its system.

The system variable should be stored in either a global or perhaps in the window's extra bytes. This variable is used when requesting or setting options for the system's dock, toolbars and items. See the source code for the demonstration application to see how this is done.

### Sample Code

```
ATCSYSTEM      pATCSys;          // The ATC system

// Install a toolbar system for this window
if ((pATCSys = ATC_InstallSystem (hWnd)) == NULL)
    return (NULL);

// Remove a toolbar system
ATC_RemoveSystem (pATCSys);
```



## Defining Buttons






First a bitmap must be defined for all the button images. This bitmap contains all the "up" or "normal" images for all the buttons. For example the following image bitmap was used for the sample program:



Each image is the standard 16 pixels wide by 15 pixels tall. After adding the appropriate shading this will create a standard 24x22 pixel button as specified by *The Windows Interface: An Application Design Guide* published by Microsoft Press. This default button size can be changed.

The images can continue for several rows allowing an almost unlimited number of buttons. The ordering of images is defined as increasing from left to right, wrapping to the next row, and continuing through all the images.

ATC will automatically modify the given image to indicate the following button states:

	up
	down
	checked
	indeterminate
	disabled

Finally, a command ID is assigned for each button in the same order as the images in the bitmap.

### Sample Code

```
// Define the command ID, quick help ID, and description ID arrays
UINT uiCmdIDArray[] = {ID_FILE_NEW, ID_FILE_OPEN, ID_FILE_SAVE,
                       ID_EDIT_CUT, ID_EDIT_COPY, ID_EDIT_PASTE,
                       ID_FILE_PRINT, ID_APP_ABOUT};

// The bitmap "Toolbar" contains all the toolbar images
ATC_DefineButtonBitmap (pATCSystem, "Toolbar");

// Then assign buttons and strings to the toolbar images
// Use the same order as bitmap images
ATC_DefineButtons (pATCSystem, uiCmdIDArray,
                  sizeof (uiCmdIDArray) / sizeof (UINT));
```

## Defining Categories

Categories may be used to separate buttons into groups for the Insert Item dialog box. This feature is easy to implement for the programmer and extremely useful for the user since the search for a particular toolbar item can be accomplished in much less time.

Note that categories are optional. If none are specified then the Categories combo box in Insert Item dialog has one entry which reads "All Items."

A categories heading array is passed to ATC and is used, as given, as the list for the Insert Item category combo box. The array is filled with the string ID's of the actual category headings such as "File" and "Edit".

The items within the categories is then passed to ATC as an array of all command ID's which make up a given category.

### Sample Code

```
// Category headings and categories
UINT uiHeadingsIDArray[] = {ID_FILE, ID_EDIT, ID_MISC};
UINT uiCatFileIDArray[]  = {ID_FILE_NEW, ID_FILE_OPEN,
                           ID_FILE_SAVE, ID_FILE_PRINT};
UINT uiCatEditIDArray[]  = {ID_EDIT_UNDO, ID_EDIT_CUT,
                           ID_EDIT_COPY, ID_EDIT_PASTE};
UINT uiCatMiscIDArray[]  = {ID_APP_ABOUT};

// Define the category order
if (!ATC_DefineCategoryOrder (pATCSystem, uiHeadingsIDArray,
                             sizeof (uiHeadingsIDArray) / sizeof (UINT))
    return (NULL);

// Define the categories
if (!ATC_DefineCategory (pATCSystem, ID_FILE, uiCatFileIDArray,
                        sizeof (uiCatFileIDArray) / sizeof (UINT))    ||
    !ATC_DefineCategory (pATCSystem, ID_EDIT, uiCatEditIDArray,
                        sizeof (uiCatEditIDArray) / sizeof (UINT))    ||
    !ATC_DefineCategory (pATCSystem, ID_MISC, uiCatMiscIDArray,
                        sizeof (uiCatMiscIDArray) / sizeof (UINT)))
    return (NULL);
```

## Defining Docks

Currently ATC only supports a top dock. To define this dock an ATC function is used and the handle to the dock window is returned. This window handle should be used when sizing child windows within the application parent window.

When the toolbars are added or removed from the dock window the application window is sent size message (WM\_SIZE) to update the display.

Note that from Microsoft Foundation Class (MFC) applications you must create the dock using a special wrapper around ATC\_CreateDock. See the MFC [example](#) code for more information.

### Sample Code

```
// Create a dock window  
hWndDock = ATC_CreateDock (pATCSystem, hWnd, ATC_DOCK_TOP);
```

## Defining Toolbars

Toolbars are defined by using a single ATC function and passing in the command IDs of all buttons to be installed on the toolbar. The identifier `ID_ATC_SEPARATOR` (defined as 0) should be used to add a separator between toolbar items.

Toolbar windows can also have special attributes such as:

- `ATC_NO_SWITCH`. Forces the toolbar to remain in its current state. It can't be switched between floating and docked modes.
- `ATC_FLOATCOMPRESSED`. When the toolbar is floating, separators ("dead" items) will be temporarily removed. This will force all the buttons to press against each other creating the familiar tool palette seen in many graphics applications.
- `ATC_NO_CONFIG`. This items in this particular toolbar cannot be changed in any way. New items cannot be inserted; existing items cannot be removed.

Several useful combinations of the available styles have been combined into common styles like `ATC_SHOWTOOLPALETTE` and `ATC_SHOWDOCKED`.

### Sample Code

```
// Define the standard toolbar
UINT uiStandardIDArray[] = {ID_FILE_NEW, ID_FILE_OPEN, ID_FILE_SAVE,
                            ID_ATC_SEPARATOR,
                            ID_EDIT_CUT, ID_EDIT_COPY, ID_EDIT_PASTE,
                            ID_ATC_SEPARATOR,
                            ID_FILE_PRINT, ID_APP_ABOUT};

// Append the "standard" toolbar to the end of the created dock
hWndToolbar = ATC_AppendToolbar ("Standard", ATC_SHOWDOCKED,
                                0, 0, 0, 0,
                                hWndDock, uiStandardIDArray,
                                sizeof (uiStandardIDArray) / sizeof (UINT))

// Append a no-config "palette" toolbar to the end of the created dock
hWndToolbar = ATC_AppendToolbar ("Standard",
                                ATC_SHOWTOOLPALETTE | ATC_NO_CONFIG,
                                0, 0, 0, 0,
                                hWndDock, uiStandardIDArray,
                                sizeof (uiStandardIDArray) / sizeof (UINT))

// Mark a window as visible, hidden, floating, then docked
// Don't use Windows' ShowWindow!!!
ATC_ShowToolbar (hWndToolbar, ATC_SHOW);
ATC_ShowToolbar (hWndToolbar, ATC_HIDE);
ATC_ShowToolbar (hWndToolbar, ATC_SHOWFLOATING);
ATC_ShowToolbar (hWndToolbar, ATC_SHOWDOCKED);
```

```
// Is toolbar visible? (don't use Windows' IsWindowVisible!)
if (ATC_IsVisible (hWndToolbar))

// Is toolbar floating?
if (ATC_IsFloating (hWndToolbar)) ...
```

## Button Styles

There are three button styles that can be used by the programmer:

1. Normal. A button is pressed down by the user and it is returned to the up position automatically by ATC. This is the default button style.
2. Checkbox. A button is pressed down by the user and it automatically changes between checked and unchecked states without returning to an up state.
3. Stay-pressed. A button is pressed down by the user and it remains down until the application releases it with an API call. This can be used as feedback for the user to indicate that the program is still working on the requested command.

### Sample Code

```
// Make the bold button a checkbox
ATC_SetButtonStyle (pATCSystem, ID_STYLE_BOLD, ATC_BTN_CHECKBOX);

// Make the about button a stay-pressed button
ATC_SetButtonStyle (pATCSystem, ID_APP_ABOUT, ATC_BTN_STAYPRESSED);
```

## Button States

ATC provides several functions to modify the state of a particular button. Buttons can be enabled or disabled and unchecked, checked, or indeterminate. Buttons can also be enabled or disabled and pressed or released.

### Sample Code

```
// Make the underline button unchecked
ATC_CheckButton (pATCSysyem, ID_STYLE_BOLD, FALSE);

// Make the italic button checked
ATC_CheckButton (pATCSysyem, ID_STYLE_BOLD, TRUE);

// Make the bold button indeterminate
ATC_CheckButton (pATCSysyem, ID_STYLE_BOLD, 2);

// The save button should be disabled
ATC_EnableButton (pATCSysyem, ID_FILE_SAVE, FALSE);

// The stay-pressed compile button should now be released
ATC_PressButton (pATCSysyem, ID_FILE_COMPILE, FALSE);

// Make the left radio button checked and turn off all others
ATC_CheckRadioButton (pATCSysyem, ID_STYLE_LEFT, ID_STYLE_JUSTIFIED,
                     ID_STYLE_LEFT);

// Is the bold button checked?
if (ATC_IsButtonChecked (pATCSysyem, ID_STYLE_BOLD))    ...

// Is the bold button enabled?
if (ATC_IsButtonEnabled (pATCSysyem, ID_STYLE_BOLD))    ...

// Is the bold button pressed?
if (ATC_IsButtonPressed (pATCSysyem, ID_STYLE_BOLD))    ...
```

## Toolbar Settings

The ATC system can be configured with the following options:

- An individual toolbar can be configured as floating or docked only (no switching).
- An individual toolbar can be configured to temporarily remove all item separators when the window is floating to create a palette window.
- An individual toolbar can be configured to be non-configurable by the user so items cannot be moved or inserted.

### Sample Code

```
// This toolbar can no longer be switched between
// floating and docked forms.
dwStyle = ATC_GetToolbarStyle (hWndToolbar);
ATC_SetToolbarStyle (hWndToolbar, dwStyle | ATC_NO_SWITCH);

// When this toolbar is floating, separators are temporarily
// removed to compress the toolbar into a traditional "palette"
// seen in graphics applications.
dwStyle = ATC_GetToolbarStyle (hWndToolbar);
ATC_SetToolbarStyle (hWndToolbar, dwStyle | ATC_FLOATCOMPRESSED);

// This toolbar can no longer be configured by the user
dwStyle = ATC_GetToolbarStyle (hWndToolbar);
ATC_SetToolbarStyle (hWndToolbar, dwStyle | ATC_NO_CONFIG);
```



## Status Bar Messages

An expected feature in Windows applications is descriptive help text in the status bar when a menu item or toolbar item is selected. To notify the application that a button is being pressed or released, ATC provides a function to define certain messages.

Two messages are required:

1. Button Select - to indicate that a button is currently being selected.
2. Button Cancel - to indicate that a button is no longer selected but was not actually chosen. This is accomplished by releasing the mouse button while the pointer is not over the toolbar item.

Note that these messages following the wParam/lParam format of WM\_MENUSELECT. So, you can request that ATC simply use WM\_MENUSELECT to indicate button presses.

If an item is selected and then released the application window will receive a WM\_COMMAND message with the command ID in the word parameter.

For compatibility with MFC-based application (and the built-in CStatusBar support) you should use ATC\_DefineMessages to mimic the WM\_MENUSELECT messages.

### Sample Code

```
// Messages from ATC to this app to update status bar text
#define WM_ATM_BUTTONSELECT      WM_USER + 1      // Button selection
#define WM_ATM_BUTTONCANCEL     WM_USER + 2      // Button cancellation

// We want to get status bar update messages
ATC_DefineMessages (WM_ATM_BUTTONSELECT, WM_ATM_BUTTONCANCEL);

// We want to get status bar update messages via WM_MENUSELECT
ATC_DefineMessages (WM_MENUSELECT, WM_MENUSELECT);
```

## Application Sizing

When the dock window is resized, the ATC library sends a WM\_SIZE message to the parent application window. The method for calculating the size taken up by the dock window is easily obtained using a single function. See the following sample code for more information. If you are using ATC with an MFC-based application then the wrapper class demonstrated in the MFC [example](#) code handles resizing automatically.

### Sample Code

```
// Get the top dock window and figure out its size
hWndDock = ATC_GetDockWindow (pATCSystem, ATC_DOCK_TOP);
if (hWndDock && IsWindowVisible (hWndDock))
{
    RECT sRectDock;

    // Use GetWindowRect (instead of GetClientRect) so we get
    // the borders of the dock window - the true height
    GetWindowRect (hWndDock, &sRectDock);
    iDockHeight = sRectDock.bottom - sRectDock.top;
}
```



```
// Free the layout strings
ATC_FreeLayoutStrings ();
}
```

## Layout Restoration

A toolbar layout can be neatly restored using the layout strings created using the layout saving routines as demonstrated in the previous section. The sample code below demonstrates a RestoreLayout function which takes the layout strings stored in the application's INI file under the given INI section and converts them to the format needed by ATC.

This function returns TRUE if all is successful. If a problem occurs or there are no layout strings in the INI file then it will return FALSE and the application should simply create the default toolbars manually.

### Sample Code

```
BOOL                                     // TRUE if restored; otherwise FALSE
RestoreLayout(
    ATCSYSTEM      pATCSys,           // The ATC system
    HWND           hWnd,             // The parent window
    LPSTR          szINISection)     // The INI file section
{
    char           rcINIFile[255];    // The INI file
    char           rcHeadings[255];  // All the headings
    BOOL           fRestore;         // TRUE if it was restored
    char           *szHeading;       // The current heading

    // The INI file name is the app name with an INI extension
    GetModuleFileName (ghInstance, rcINIFile, sizeof (rcINIFile) - 1);
    lstrcpy (rcINIFile + strlen (rcINIFile) - 3, "INI");

    // Get all the headings in the INI file under the layout section
    // Just return with FALSE if no layout saved
    if (!GetPrivateProfileString (szINISection, NULL, "",
                                  rcHeadings, 255, rcINIFile))
        return (FALSE);

    // Point to the top of the headings list
    // This list of headings is \0 separated with a \0\0 at the end
    szHeading = rcHeadings;

    // Init the layout string arrays
    ATC_InitRestoreLayout ();

    // Get all the layout strings from the INI file
    while (szHeading[0] != '\0')
    {
        char           rcLayout[255]; // Heading layout string
```

```
    // Using the current heading, get it's information
    if (GetPrivateProfileString (szINISection, szHeading, "",
                                rcLayout, 255, rcINIFile))
        if (!ATC_AddLayoutString (szHeading, rcLayout))
            return (FALSE);

    // Skip down to the next heading
    szHeading += lstrlen (szHeading) + 1;
}

// Do the restoration then free the layout strings
fRestore = ATC_RestoreLayout (pATCSys, hWnd);
ATC_FreeLayoutStrings ();
return (fRestore);
}
```

## "Toolbars" Menu

Adding a "Toolbars" menu to your application allows users to show or hide toolbars through a keyboard interface. It also serves another important purpose: it is actually possible for the user to hide all the toolbars and have no way to get them back! This can happen if the user hides all the toolbars using the ATC-supported [popup toolbar menu](#). The user can no longer click on a dock or toolbar to access the popup toolbar menu! Therefore, a "Toolbars" menu is extremely important.

The easiest way is shown in the sample applications and is demonstrated in the example code below. Use `CreatePopupMenu` and `AppendMenu` to add a "Toolbars" popup menu to your application window's menu structure. On the `WM_INITMENUPOPUP` message for that window update the "Toolbars" menu using the `UpdateToolbarsMenu` function shown below. Then on a `WM_COMMAND` message for one of the items in that menu use ATC to find and show the selected toolbar window.

### Sample Code

```
void
UpdateToolbarsMenu(
    ATCSYSTEM    pATCSys,          // The ATC system
    HMENU        hMenu)           // The "Toolbar" submenu
{
    UINT         uiNumToolbars;    // # toolbars in system list
    UINT         uiIndex;         // Index into toolbar list

    // Remove items if any items exist
    uiNumToolbars = GetMenuItemCount (hMenu);
    for (uiIndex = 0; uiIndex < uiNumToolbars; uiIndex++)
        DeleteMenu (hMenu, 0, MF_BYPOSITION);

    // Get the toolbar count, if zero then return
    if ((uiNumToolbars = ATC_GetToolbarCount (pATCSys)) == 0)
        return;

    // Loop through the toolbar list
    for (uiIndex = 0; uiIndex < uiNumToolbars; uiIndex++)
    {
        HWND      hWnd;           // The toolbar window

        // Get the toolbar window
        if ((hWnd = ATC_GetToolbarViaIndex (pATCSys, uiIndex))
            != NULL)
        {
            char    rcTitle[255];  // Title of the window
            BOOL    fVisible;     // TRUE if window is visible
        }
    }
}
```

```
    // Get the title and visibility of the window
    ATC_GetToolbarTitle (hWnd, rcTitle, sizeof (rcTitle) - 1);
    fVisible = ATC_IsVisible (hWnd);

    // Append the toolbar to the menu
    AppendMenu (hMenu, MF_ENABLED | (fVisible ? MF_CHECKED : MF_UNCHECKED),
               ID_TBARSTART + uiIndex, rcTitle);
}
}
}
```



## MFC Support

The example MFC application included with the ATC installation demonstrates the use of ATC and MFC within the same program. Please follow the following instructions with those source files.

Changes primarily involve the `MAINFRM.CPP` and `MAINFRM.H` files. You will also have to include two new files which serve as an MFC wrapper around ATC.

Here are the important points to keep in mind:

1. Include the `ATCWRAPR.CPP` and `ATCWRAPR.H` files into your project. These files define a very simple class called `CATCWrapper` and can be found in the `MFCDEMO` directory. This class wraps around the dock window generated by ATC.
2. Remove all references to the MFC `CCToolBar` control in `MAINFRM.CPP` and `MAINFRM.H`.
3. Add references to the `CMainFrame` class (in `MAINFRM.H`) for the public `CreateToolBar` function and the protected `m_wndATCWrapper` variable.
4. `#include "atcwrapr.h"` in your `MAINFRM.CPP` file.
5. Initialize ATC in the `CMainFrame::OnCreate` handler, also call `CreateToolBar` to create the actually ATC dock and toolbar. If you use `ATC_DefineMessages` to mimic `WM_MENUSELECT` then ATC will work with the MFC status bar.
6. You will have to supply a `ATCLoadStringProc` function to grab description strings.
7. Follow the sample `CreateToolBar` function to see exactly what is necessary to install an ATC toolbar. Note that the technique is exactly the same as what's necessary for a non-MFC application with one exception. Use the `m_wndATCWrapper.CreateDock` member function instead of `ATC_CreateDock`.
8. Add the `ATC16.LIB` library file to the link options for your project. Remember that the `ATC16.DLL` must be in the same directory as your application during run-time.

At some point you will probably want to define quick help text. This will involve changes to your RC file. See the `MFCDEMO` and `ATCDEMO` files for examples.

## **Object-Oriented Design**

The ATC library was carefully designed and implemented so future upgrades and features would be easy to add. While the source code is written in C for compatibility with SDK applications, object-oriented techniques were used throughout the code to insure maximum portability and data abstraction.

The code that manipulates the internal toolbar data is "black-boxed" within several code modules. Access to data structure variables is only permitted through API calls. These data structures can be extensively modified without affecting external code modules.

The toolbar handles items - not buttons. Currently there are button items and dead items (used for separators). In the future other items such as those defined by a window class (such as combobox or a user-defined clock window class) can be included on a toolbar.

All routines which deal directly with the Windows API are in separate independent modules. This leads to the possibility of porting to other operating environments or taking advantage of new Windows API calls in the future.

All routines which deal directly with memory are in a separate module. If the application currently uses a private memory management scheme, the ATC library can use it as well.

## Robust Source

The code for the ATC library was compiled using Microsoft Visual C++ Version 1.5 at the highest warning level (-W4). Warnings for the use of // for comments and unused formal parameters are the only two warning types that were ignored. The STRICT definition was also used to guarantee correct Windows API function calls.

The code also compiles cleanly under Borland C++ 4.0. The same warnings accepted from the Microsoft compiler were accepted from Borland.

Nu-Mega's Bounds Checker 2.0 for Windows was used for extensive testing of the ATC library. No memory leaks or illegal Windows calls were reported.

The library has also been tested under the debugging version of Windows 3.1 and executes cleanly.

The 16-bit library itself is compiled using the large memory model. However, the DLL works perfectly with mixed-model applications (small, medium, or large memory models).

As stated earlier, the ATC library was completely written in the C programming language. This allows maximum source code compatibility with C or C++ applications. The header file used by the application to access the DLL, ATC.H, has been delimited with the extern "c" instruction to work with C++ source code.

The 16-bit and 32-bit ATC libraries are based on the same source code. The migration of the ATC source code to the Win32 forced even cleaner code. Special care was taken to insure source code compatibility between 16-bit and 32-bit environments.

With over 15,000 lines of heavily documented code, the source also provides excellent examples of the following high-level Windows operations:

- Manually drawing graphical buttons (not simply ownerdraw buttons)
- Popup menus for the right mouse button
- Converting child windows into popup windows at run-time
- Manually drawing mini title bars
- Subclassing a listbox to support drag and drop
- Ownerdraw, variable-height graphical listboxes
- Timers
- Constructing modules which compile for 16- and 32-bit Windows libraries

## Ordering Information

*The Advanced Toolbar Control is available now!*

ATC is priced based on the licensing and source code options listed below:

1. 16-Bit Libraries for \$195
2. 16/32-Bit Libraries for \$295
3. 16/32-Bit Libraries and Source Code for \$495

There are no royalties to pay after purchasing the ATC libraries. The libraries are to be used on a per-product basis; to develop additional software products additional licenses must be purchased. For more information read the [licensing agreement](#) included with this help file.

To make update announcements easier please provide us with an email address and a list of on-line services you frequent.

To order, please send a check made out to *Alien Technologies* to the following address. Or call us with your Mastercard or Visa number. North Carolina residents are required to add 6% to the package price for sales tax. Shipping and handling charges are \$5 for 3-day arrival and \$10 for next-day air within the continental United States. For other destinations please contact us at the phone number listed below.

Alien Technologies  
3330 Walnut Creek Parkway Suite N  
Raleigh, NC 27606-3840  
(919) 851-9622

Internet: [aliensales@aol.com](mailto:aliensales@aol.com)  
America Online: AlienSales  
CompuServe: 74367,3261

## **16-Bit Libraries**

This package consists of the files necessary to integrate ATC into Windows 3.1 applications. Sample code is provided which demonstrate how to use ATC with C/SDK applications and with C++/MFC applications. Project files are supplied for use with Microsoft or Borland compilers.

The complete API reference is supplied within an extensive Windows help file. Functions are hypertext-linked to example code.

## **16/32-Bit Libraries**

This package consists of the files necessary to integrate ATC into Windows 3.1 and Windows NT applications. Sample code is provided which demonstrate how to use ATC with C/SDK applications and with C++/MFC applications. Project files are supplied for use with Microsoft or Borland compilers.

The complete API reference is supplied within an extensive Windows help file. Functions are hypertext-linked to example code.

## **16/32-Bit Libraries and Source Code**

This package consists of the files necessary to integrate ATC into Windows 3.1 and Windows NT applications. Sample code is provided which demonstrate how to use ATC with C/SDK applications and with C++/MFC applications. Project files are supplied for use with Microsoft or Borland compilers.

The complete API reference is supplied within an extensive Windows help file. Functions are hypertext-linked to example code.

In addition, the complete source code to the ATC libraries are provided. This code can be compiled using Microsoft or Borland compilers in 16- or 32-bit environments.

The licensing agreement allows you to modify the source code and create new, personalized versions of ATC that you can distribute, in object code form, with your application.

You are not permitted to use the source code to create a product with would directly compete with the ATC toolbar control product.

# Licensing Agreement

## ADVANCED TOOLBAR CONTROL

This is a legal agreement between you (either an individual or an entity) and Alien Technologies. By installing the computer software in this package ("SOFTWARE"), by loading or running the software, or by placing or copying the files contained on this disk you are agreeing to be bound by the terms of this Agreement. If you do not agree to the terms of this Agreement, promptly return the SOFTWARE and the accompanying items (including all written materials), along with your receipt to the place from where you obtained them for a full refund.

## ALIEN TECHNOLOGIES LICENSE AGREEMENT

1. **GRANT OF LICENSE.** Alien Technologies grants to you the non-exclusive rights to use the enclosed software and electronic text files comprising the documentation (the "SOFTWARE") in the following manner. You may use the SOFTWARE for the development of one software product; additional licenses can be purchased from Alien Technologies to develop additional software products. You may make an unlimited number of copies of any material copied from the SOFTWARE or from other written materials accompanying the SOFTWARE, provided that such copies shall be used only for internal purposes and are not to be republished or distributed (either in hardcopy or electronic form) beyond your premises, except as otherwise provided herein.
2. **COPYRIGHT.** The SOFTWARE is owned by Alien Technologies or its suppliers and is protected by United States copyright laws and international treaty provisions. You may not otherwise reproduce, copy, disclose to others, or distribute, in whole or in part, the SOFTWARE except as otherwise provided herein.
3. **OTHER RESTRICTIONS.** You may not rent or lease the SOFTWARE. You may make a permanent transfer of the SOFTWARE by providing Alien Technologies written notice of your name, company, and address and the name, company, and address or the person to whom you are transferring the rights granted within. In the event of a transfer, you may not retain any copies of the SOFTWARE and accompanying written materials, and the recipient must agree to the terms of this Agreement. You may not reverse engineer, decompile, or disassemble the SOFTWARE. If the SOFTWARE is an update or has been updated, any transfer of the license to use the SOFTWARE must include the most recent update and all prior versions.
4. **SAMPLE CODE.** In addition to the rights granted in Section 1, Alien Technologies grants you a non-exclusive, royalty-free right to use and modify the source code located within the "samples" directory (Sample Code) for the sole purposes of designing, developing, and testing your software product, and to reproduce and distribute the Sample Code along with any modifications thereof, only in object code form provided that you comply with Section 7.
5. **SOURCE CODE.** In addition to the rights granted in Section 1, if you have purchased the Source Code to this product, Alien Technologies grants you a non-exclusive, royalty-free right to use and modify the source code located within the "source" directories (Source Code) for the sole purposes of designing, developing, and testing your software product, and to reproduce and distribute the Source Code along with any modifications thereof, only in object code form provided that you comply with Section 7.
6. **REDISTRIBUTABLE COMPONENTS.** In addition to the rights granted in Section 1, Alien Technologies grants you a non-exclusive, royalty-free right to use the object code files located in the "redist" directory (Redistributable Code) for the sole purposes of designing, developing, and testing your software product, and to reproduce and distribute the Redistributable Code only in object code form provided that you comply with Section 7.
7. **REDISTRUBUTABLE REQUIREMENTS.** If you are authorized to redistribute the Sample Code and/or Redistributable Code, (collectively "REDISTRIBUTABLE COMPONENTS") as described in Sections 4, 5 and 6 above, you must: (a) distribute the REDISTRIBUTABLE COMPONENTS only in conjunction with and as a part of your software application product; (b) not permit further redistribution of the REDISTRIBUTABLE COMPONENTS by your end-user customers; (c) not use Alien Technologies' name, logo, or trademarks to market your software application product; (d) include a valid copyright notice on your software application product; and (e) agree to indemnify, hold harmless, and defend Alien Technologies from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of your software application product.



#### LIMITED WARRANTY

LIMITED WARRANTY. Except with respect to the SAMPLE CODE which is provided "as is," without warranty of any kind, Alien Technologies warrants that (a) the SOFTWARE will perform substantially in accordance with the accompanying written or on-line documentation for a period of ninety (90) days from the date of receipt. Any implied warranties on the SOFTWARE is limited to ninety (90) days. Some states/jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you.

CUSTOMER REMEDIES. Alien Technologies' and its suppliers' entire liability and your exclusive remedy shall be, at Alien Technologies' option, either (a) return of the price paid, or (b) repair or replacement of the SOFTWARE that does not meet Alien Technologies' Limited Warranty and which is returned to Alien Technologies with a copy of your receipt. The Limited Warranty is void if value of the SOFTWARE resulted from accident, abuse, or misapplication. Any replacement SOFTWARE will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

NO OTHER WARRANTIES. To the maximum extent permitted by law, Alien Technologies disclaims all warranties, either express or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose and any warranty against infringement, with regard to the SOFTWARE and any accompanying written materials. This limited warranty gives you specific legal rights. You may have others which vary from state/jurisdiction to state/jurisdiction.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. To the maximum extent permitted by the applicable law, in no event shall Alien Technologies or its suppliers be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use the SOFTWARE, even if Alien Technologies has been advised of the possibility of such damages. Because some states/jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

#### U.S. GOVERNMENT RESTRICTED RIGHTS

The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights 48 CFR 52.227-19, as applicable. Manufacturer is Alien Technologies, 2522 Clark Avenue, Raleigh NC 27607.

If you acquired this product in the United States, this Agreement is governed by the laws of the State of North Carolina. If you acquired this product outside the United States, local law may apply

## **About Alien Technologies**

*Who (or what) is Alien Technologies?*

Alien Technologies was founded to provide developer and application components for the Microsoft Windows community. By components we mean independent modules that extend the capabilities of commercial applications. Components might be installed at run-time by the end-user or at compile-time by the application developer. We are very interested in your comments about this product or ideas for future products.

Please feel free to contact us with questions, comments, or orders at the following address.

Alien Technologies  
3330 Walnut Creek Parkway Suite N  
Raleigh, NC 27606-3840  
(919) 851-9622

Internet: [aliensales@aol.com](mailto:aliensales@aol.com)  
America Online: AlienSales  
CompuServe: 74367,3261

