Bitwise Operator

by Matt Slot

This month we will be discussing two parts of the applications interface: menus and windows. In addition to an overview of these objects, updated source code (found at http://www.ambrosiasw.com/~fprefect/bitwise/issue4/bitwise.c)illustrates a working implementation. As always, we'll be be throwing around many Toolbox utility functions. Remember that the Inside Macintosh books (or a copy of THINK Reference) are your friends -- take the time to look up those things that confuse you.

As a preface to the expanded code, I'd like to briefly discuss the convention for function names that it uses. Functions can be divided into 4 categories: actions, handlers, accessors, and utilities. Action functions typically perform a specific action, such as initializing the program or creating a window. You can identify an action function because its name begins with DoXXX().

Handler functions are called in response to a user or system event, to get more information and the call the appropriate action routines in response. All handler functions are of the form HandleXXX(). Accessor functions are used to insert or extract data from an interface, and typically use the GetXXX() or SetXXX() conventions. Finally, miscellaneous utility functions that don't fit into any of the above categories tend to have simple descriptive names with no clever prefix.

User Interface: Menus

The first thing we need to do is create some menus and place them into the menubar when the application starts up. DoInitialize() invokes CreateMenus() after the Toolbox has been initialized to do this. This function uses the menu and item constants to create each menu, fill them with items, and insert them one at a time into the menu bar. After each menu has been inserted, the function forces the menu bar to redraw with its new contents.

Most items have both text and a "command-key shortcut" to help the user select items from the keyboard. Once the menu is created, each item is inserted and the command key assigned. The magic "\p(-" is used to insert a disabled separator item. Be sure to avoid marks of punctuation in menu items (other than periods or ellipses), because functions like AppendMenu() may interpret them in unexpected ways.

The Apple menu is a special case, because it needs to be filled with a list of installed Desk Accessories -- or more
appropriately under System 7.0 and later, list of the user's Apple Menu Items. As illustrated in the sample, this is performed by the function AppendResMenu() with the indicated parameters (for historic reasons). When an item is chosen from the Apple menu, the text of that item must be

passed to the Toolbox routine OpenDeskAcc().

In the upcoming sections we'll discuss how the application interacts with the user via Toolbox functions, and how it handles the sample commands we've made available. Finally, during the application cleanup most programs don't bother removing and release installed menus -- since the application's memory is entirely recycled.

User Interface: Windows

The most important elements of a user interface are the windows. These are usually created and disposed by the user, either directly or indirectly. In the sample application, a window is created in response to the New item from the File menu using the NewWindow() Toolbox function. The code illustrates how to specify different attributes for the new window, and even uses the global gWindowOffset to stack subsequent windows nicely.

When the user no longer wishes to use the window, he can either select Close from the File menu or click in the "go away" box in the window's title bar. The application also needs to close each window just before quitting, so that it can offer the user a chance to save any open documents. In each case, for now, we simply locate the appropriate window call DisposeWindow().

Between creating and deletion, the application may display many windows on the screen. Normally a programmer likes to track every allocated structure manually (such as in a global or linked list), but that's not necessary under the MacOS. The Window Manager provides several ways to identify the window that requires processing, and offers a way for the application to attach private data to a given window (such as preferences, text, etc.).

If the application must iterate the windows, it can use the FrontWindow() call to find the frontmost visible window and then use the nextWindow field of the WindowRecord to continue down the linked list. By typecasting a pointer into the window reference constant using SetWRefCon() and GetWRefCon(), the application can store a private data structure that identifies the purpose and contents of a given window.

Interacting with the User

Now that we have a few user interface elements onscreen, we need to let the user play with them. As discussed in the previous column, the application event loop waits for user events and then dispatches them to be handled. In this column, we begin to fill out the handler code as it relates to windows and menus.

The most first event that an application may receive is a mouseDown event, however a click has many different semantics. The program must call the Toolbox function FindWindow() to determine which onscreen object was clicked, and then further use that information to resolve the user event. For example part code inDesk indicates that the user clicked outside of your application and there is no action necessary.

When the user clicks in the menu bar, call the Toolbox function MenuSelect() to track the mouse at it passes over menus then return which menu and item was picked. In the sample application, we extract the individual menu and item IDs and pass them to HandleMenuSelection() which takes the appropriate action.

When the user clicks in a window, the application has to react to the clicked window part by either activating, dragging, or resizing the window. Clicks in the content area are often used to select text or activate controls, but the sample source simply displays a graphic effect for the time being.

Note that clicks in the window's go away, or close, box and the zoom box involve an extra step. The user may click inside the part, but then change his mind and drag out before releasing. Use TrackGoAway() or TrackBox() to wait for the user to release the mouse before performing the indicated action. Its also important to remember to set the window's drawing port using GetPort() and SetPort() before calling ZoomWindow() or the application will crash.

The next user event that the application may receive is a keypress. If the user has pressed a key while the command modifier key is pressed, we must call MenuKey() to determine which menu item was selected and pass that information to the application's HandleMenuSelection() function. Otherwise normal keyDowns, and all autoKey events are passed to our HandleKeydown() function -- which currently just flashes the frontmost window to show that it works.

Finally, the application may receive events in response to changes in the window ordering or location on the screen. The pointer to the window is placed in the event record's message (and must be typecasted to extract), to indicate which object to handle.

In response to an updateEvt, the application must inform the Toolbox that it will be redrawing the contents of the window by wrapping the drawing operations with the functions BeginUpdate() and EndUpdate(). For now, we simply draw the grow icon and fill the window with a harmless pattern.

In response to an activateEvt, the application would normally adjust the focus of objects in the window (like text or lists) by dimming or enabling them. Until the sample windows actually contain something, we don't bother doing anything.

Overall, the process of interacting with the user is pretty simple if tedious. Each possible event, each possible item that can be clicked must be anticipated and handled. Many window parts, as well as controls, lists, and text fields have functions which help determine where and how to handle the user's actions, and another set of functions to actually perform them. The application's responsibility is to receive the events and dispatch them.

Matt Slot, Bitwise Operator