Menu Verify Handling

by John Orr and Peter Cherna

Menu verify has been a part of Intuition for a long time, so it may seem a bit odd to see an Amiga Mail article on the subject. Unfortunately, the ‘‘Intuition Menus’’ chapter of the *Amiga ROM Kernel Reference Manual: Libraries* is a bit unclear on the subject. It does not sufficiently cover why an application would need to use this feature and how an application should handle it.

Menu Verify

When the user tries to pull down the active window’s menu, Intuition offers applications a chance to prepare for the menu operation. Intuition delays displaying any menus until the application has finished its preparations for the menu operation. This feature is known as *menu verify*.

Menu verification works using the IDCMP mechanism. For an Intuition screen, when the user triggers Intuition into drawing the menu, Intuition sends an `IDCMP_MENUVERIFY` message to every window on that screen that asked to hear about menu verify events. Intuition will not display the menus until it gets back all the `IDCMP_MENUVERIFY` messages.

If the active window uses menu verification, Intuition gives that window the option to cancel the menu operation. This allows the application to selectively use the menu button to activate the menu or to perform some other function.

Beginning with V36, Intuition introduced a time out on menu verify operations. If the active window’s application takes too long to respond to the `IDCMP_MENUVERIFY` message, Intuition will cancel the menu operation. This feature helps avoid some deadlocks that can occur while Intuition is waiting on the active window. This feature applies only to the active window; Intuition will wait indefinitely for the `IDCMP_MENUVERIFY` message from an inactive window.

Reasons to Use Menu Verify

The primary purpose of the menu verify feature is to arbitrate access to screen resources. There are really only two screen resources that an application might need to use directly, the palette and the bitmap.

Some applications need as many pens as possible, so they change the color values of the pens that Intuition uses to render the menus. This can present a problem if the application happens to alter the menu pen colors so that the menu's foreground color is difficult (if not impossible) to see against the background color. The menu items will be unreadable to the user.

To prevent this problem, the application has to restore the original pen colors when the user pulls down the menu. The application can do this using menu verify. When the user hits the menu button, Intuition sends the application a menu verify message. When the application receives this message, it restores the menu colors and then returns the `IDCMP_MENUVERIFY` message so Intuition can properly render the menus. When the menu operation is over, Intuition will send a ‘‘conclusion’’ IDCMP message. The ‘‘conclusion’’ message is covered later in this article.

Another reason to use menu verify is to arbitrate rendering access to an Intuition Screen's bitmap. While Intuition is displaying the menus, the application must not render directly to the screen's bitmap because the application could draw over the menus. The application must wait until the menu operation is over to resume rendering.

This raises some important questions. Why render into a screen's bitmap (Screen->RastPort.BitMap)? Shouldn't I render into a window instead?

The quick answer is: ‘‘render into a window's rastport instead of rendering directly into a screen's bitmap’’. Rendering to a window's rastport offers several advantages over rendering directly to the screen's bitmap. The window offers the application efficient clipping, making the application simpler because it doesn't need to worry about rendering into rectangular bounds. The window also supports a layered display, so the application can ignore the consequences of arbitrating access to the screen's bitmap.

There are some cases where, arguably, it might be advantageous for an application to render directly into a screen's bitmap. Rendering into a screen's bitmap versus rendering to a window's rastport offers a slight improvement in performance. Part of this improvement is due to the lack of clipping on a screen's bitmap, which the window provides, so the performance improvement is questionable if you still need to perform clipping.

One case where an application has to render into a screen's bitmap is when graphics functions that operate below the layers level. For example blitter objects (BOBs) operate below the layers level. If the application is operating below the layers level, it can't render to a windowed display.

Reasons Not to Use Menu Verify

One feature of Intuition menus is that the user can toggle a menu item between a selected ‘‘checked’’ state and an unselected ‘‘unchecked’’ state. Intuition keeps track of the state, so the application only has to read the state from the MenuItem structure. In some cases, the state of the menu item can be affected by something besides the user performing menu operations, so the application has to explicitly change the state of the menu item.

Some applications use the menu verify feature as a mechanism for updating the state of the menu items. When such an application receives the `IDCMP_MENUVERIFY` message, it updates the checked flag of the menu items and then returns the menu verify message. Some applications also use this method to manage the enabled and disabled state of a menu item.

Although this method works, it should not be necessary to interrupt the menu event. Almost all applications should be able to keep their menu states current using ResetMenuStrip(). It is also possible to use SetMenuStrip(), but since ResetMenuStrip() is considerably faster than SetMenuStrip(), ResetMenuStrip() is better. See the Autodoc for ResetMenuStrip() for more details.

Some people may think one potential use for menu verify is to assign an alternative meaning to the right mouse button. For example, when the pointer is not within the title bar of a screen, some paint packages use the right mouse button to paint using the background color. Using menu verify, it is possible for a application to intercept the menu verify event, checking to see if the pointer is within the title bar. If it isn’t, the application cancels the menu event. After cancelling the menu event, Intuition will send a right button down event, which the application can interpret.

This mechanism may work, but it is not very efficient. There is a better way to do this using `WFLG_RMBTRAP`. This flag is from the Flags field in the Window structure. When set, this flag disables menu operations for the window. When the user hits the menu button, Intuition sends the window `IDCMP_MOUSEBUTTONS` events instead of menu events.

To use `WFLG_RMBTRAP` as an alternative to menu verify, the application tracks mouse movement using `IDCMP_MOUSEMOVE`. While the pointer is outside of the screen’s title bar, the application makes sure the `WFLG_RMBTRAP` flag is set. If the user hits the right mouse button while `WFLG_RMBTRAP` is set, Intuition sends the application a mouse button event. While the pointer is inside the screen’s title bar, the application makes sure the `WFLG_RMBTRAP` flag is clear. If the user hits the right mouse button while `WFLG_RMBTRAP` is clear, Intuition generates a normal menu event for the window.

`WFLG_RMBTRAP` is an exception to most fields in Intuition structures because it is legal for an application to directly modify this flag. Note that this change must take place as an atomic operation so that Exec cannot perform a task switch in the middle of the change. If you are unsure your compiler will do this, use a Forbid()/Permit() pair to prevent a task switch.

There are cases where an application is too busy to handle any menu event so it needs to prevent menu operations for a while. This is another case where menu verify is inappropriate. A better way to block menus is to remove the menu with the Intuition function ClearMenuStrip() and restore it later with ResetMenuStrip(). Another potentially useful way to block menu operations is using a requester.

Will I Ever Need to Cancel a Menu Operation?

When the active window receives an `IDCMP_MENUVERIFY` message, it has the option of cancelling the menu operation. Although this feature may have some possible uses, for the moment there is no recommended use for it.

Handling Menu Verify Events

Typically, an application uses menu verify because the application needs to know when the menus are down. This goes for the menus of any window on the screen, not just the application's window. When the application receives its `IDCMP_MENUVERIFY` message, it performs some minor processing (restoring menu pen colors, for example) and records the fact that the display is in a ‘‘menu processing’’ state (i.e. Intuition is going to display the menus). The application then returns the menu verify message so Intuition can continue with the menu operation. The application cannot utilize the screen's resources until the menu processing state is over.

There are two types of events that can end the menu processing state. A cancellation or a menu pick can terminate the menu processing state.

There are several events that can cancel a menu operation. Some possibilities include Intuition timing out waiting for an outstanding `IDCMP_MENUVERIFY`, the active window cancelling the menu operation, and Intuition failing to allocate enough resources to display the menus. The ‘‘Intuition Menus’’ chapter of the *Amiga ROM Kernel Manual: Libraries* attempts to interpret these causes based on the IDCMP messages that arrive after the application returns the `IDCMP_MENUVERIFY` message.

The book should not have mentioned how to interpret the cause of a menu cancellation. The reasoning for the cancellation is not really relevant. Trying to interpret the cause of a menu cancellation is a bad idea (especially if you interpret based on the information in the book; it's wrong). The only important fact is that the ‘‘menu processing’’ state has terminated and the application can resume normal operations.

Any of the following message combinations signify the cancellation of the ‘‘menu processing’’ state:

- IDCMP_MOUSEBUTTONS (IntuiMessage.Code is equal to MENUUP) only
- IDCMP_MENUPICK (IntuiMessage.Code is equal to MENUNULL) only
- IDCMP_MOUSEBUTTONS (IntuiMessage.Code is equal to MENUUP) then IDCMP_MENUPICK (IntuiMessage.Code is equal to MENUNULL)
- IDCMP_MENUPICK (IntuiMessage.Code is equal to MENUNULL) then IDCMP_MOUSEBUTTONS (IntuiMessage.Code is equal to MENUUP)

Since the active window can cancel menu operations, the application may need to determine if its window is the active window. The application can do this by examining the IntuiMessage.Code field of the menu verify message. If that field is equal to MENUWAITING (defined in <*intuition/intuition.h*>) the window is not active. However, if that field is equal to MENUHOT (defined in <*intuition/intuition.h*>) the window is active and can cancel the menu operation by changing the Code field of the menu verify message to MENUCANCEL (also defined in <*intuition/intuition.h*>). If the application cancels the menu operation, Intuition will send the active window an IDCMP_MOUSEBUTTONS event about the right mouse button.

The following are possible cases signifying that the ‘‘menu processing’’ state is over because the user successfully pulled down and released the menu (there was no cancellation):

- IDCMP_MENUPICK (IntuiMessage.Code is equal to the menu code)
- (spurious) IDCMP_MOUSEBUTTONS (IntuiMessage.Code is equal to MENUUP) then IDCMP_MENUPICK (IntuiMessage.Code is equal to the menu code)

Note that the menu code could be MENUNULL if no item was selected. IDCMP_MENUHELP events have the same meaning as the IDCMP_MENUPICK messages when it comes to terminating the ‘‘menu processing’’ state.

There are rare cases in 3.0 and prior, where Intuition does not send the application an IDCMP message after the user picks a menu item. The result is the application does not know that the ‘‘menu processing’’ state is over. The only notable case is when the user types a menu keyboard shortcut, and Intuition times out waiting for the application to respond to the menu verify message.

V40 should fix the lack of a terminating event in those rare cases, as well as eliminate the spurious IDCMP_MOUSEBUTTONS event that can occur for successful menu picks.

About Double Menu Requesters

Intuition has a feature that allows an application to put up a special type of requester when the user double clicks the right mouse button. This requester is known as a *Double Menu Requester*. Unfortunately, the double menu requester is considered to be inconsistant with the Amiga's user interface, so using them in any capacity is strongly discouraged. Using them in conjunction with menu verify is even more strongly discouraged because they significantly complicate processing menu verify events. *For this reason, never use double menu requester and menu verify together!*