Making Your Windows More Shiny and Manageable

By Ewout Walraven

The Intuition window paradigm has features that make it easy for the user to manage windows. The user can select, move, resize, close, or depth arrange their windows on any screen so they can quickly get to the window they need. With 2.0 and a little work, the tools are there to give the user even greater powers to manage and organize their working environment.

This article addresses some release 2.0 system features which can further integrate your application's window into the 2.0 environment. These features include public screens, the zoom gadget, AppIcons, and AppMenuItems, which are not discussed in detail here. For more information on public screens and the zoom gadget, see the *intuition.library* Autodocs and the article ‘‘An Introduction to V36 Screens and Windows’’ from the September/October issue of *Amiga Mail.* For more information on AppIcons and AppMenuItems, see the *workbench.library* Autodocs and the article ‘‘AppWindows, AppIcons, and AppMenuItems’’, which is in this issue of *Amiga Mail*.

Public Screens

The public screens introduced in 2.0 make it possible for the user to open several screens where different applications can open windows. If an application allows the user to choose the public screen on which it opens its window, the user can more easily organize their working environment. They can spread their work across several public screens allowing them to use the screen that has the palette or resolution best suited to their purposes. This also buys the user more room for windows. In case one screen gets too cluttered with windows, the user can open an application's window on a different public screen.

There are number of ways you can support public screens. If your application uses command line options or reads tooltypes, it can support the `PUBSCREEN` keyword as a command line argument or tooltype. This lets the user indicate the name of the public screen on which your windows should open. For example, if some developer wrote a new shell that supported public screens, the shell's icon could have the following tooltype in its icon:

`PUBSCREEN=`<*pubscreenname*>

which tells the shell to open its window on the public screen named *pubscreenname*.

If an application supports ARexx, it can support a `JUMPTOSCREEN` command that tells it to move, or jump, its window to a specific public screen. The application would open a new window on the public screen named in the `JUMPTOSCREEN` command and closes its old window. Your application can even provide a jump-to-next-screen gadget, that makes the window jump to the next public screen. By using the *intuition.library* NextPubScreen(), your application can get the name of the next screen in the public screen list, close its window on the current screen and re-open it on the next.

The example program *jumpy.c* opens a window which jumps from one screen to another when the user clicks the jump gadget.

Zooming

A very simple, but convenient feature you can ask the system to add to your window is a zoom gadget. A zoom gadget allows the user to toggle between two window sizes simply by clicking on a gadget. In 2.0, a window gets a zoom gadget automatically when it has both a sizing and depth gadget, although you don't need a sizing gadget to get a zoom gadget. You can ask for a zoom gadget by specifying the `WA_Zoom` tag when opening the window. The `WA_Zoom` tag takes an array of four WORDs, describing the initial LeftEdge, TopEdge, Width, and Height values of the window's alternate dimensions and position. Because such a window has no sizing gadget, the user can only toggle between two window sizes, the original size and the alternate size specified with the `WA_Zoom` tag. The preference editors use this feature to toggle between a full-sized window and a window displaying only the title bar. The example *zoom.c* shows how easy it is to implement this feature.

Iconifying

Although the zoom gadget provides a pseudo-iconifying feature, the Workbench's AppIcon and AppMenuItem features provide true iconification. To iconify a window, create the AppIcon or AppMenuItem that will replace the window. Next, store the state of the window (like its current position or the state of its gadgets), so the application can restore the window to its original state when the application reopens it. Now close the window. Because it is possible for the AddAppIcon() or AddAppMenuItem() function to fail, It is important to create the AppIcon or AppMenuItem before closing the window. If an application closes the window first and Workbench can't create the AppIcon or AppMenuItem, there will be no AppIcon or AppMenuItem for the user to select to uniconify the window.

An application can set up a gadget or menu item so the user can tell the application to iconify the window. If the application uses a gadget to iconify, it should not put the gadget in the window border because there is no standard ‘‘look’’ to what an iconify gadget should look like.

The last example, *hide.c*, shows how to use AppIcons and AppMenuItems to iconify a window.