AppWindows, AppIcons, and AppMenuItems

By Fred Mitchell and John Orr

Since its inception, the Workbench™ has had a limitation. Although it is a fairly powerful user interface, that power is not accessible to application programs. The power is limited to an interface that only launches other programs. After Workbench launches a program, the program no longer has any ties to the Workbench GUI. If an application needs an iconic interface, it has to create its own, independent of Workbench.

Workbench 2.0 is different. Through the *workbench.library*, applications can utilize the iconic interface of Workbench 2.0. There are three elements to this interface: *AppWindows*, *AppIcons*, and *AppMenuItems*. In this article, they are referred to as *AppObjects*.

When the user drops a Workbench icon onto a special kind of application window called an *AppWindow*, Workbench sends a message to the application that created the AppWindow. This message contains a complete list of the icons that the user dropped on the window. This is useful for an application like an editor. The editor can open an Intuition window on the Workbench screen and make it into an AppWindow so that when the user drops an icon on the AppWindow, the editor will load the icon's corresponding file. The *IconEdit* utility that comes on the 2.0 release disks does this.

An application can also create its own icons for the Workbench window. These icons are called *AppIcons*. They are similar to AppWindows in that Workbench will tell an application what icons the user dropped on its AppIcon. In addition, Workbench will notify the application if the user double-clicks the AppIcon. This makes AppIcons useful not only as a ‘‘drop box’’ (like an AppWindow), but they can also be used as some sort of activator for an application. For example, a word processor that opens a window on the Workbench can use an AppIcon to ‘‘iconify’’ its window. When the user wants to get rid of a cumbersome window, he iconifies it, which gets rid of the window and leaves an AppIcon on the Workbench window in its place. When the user wants the window back, he double-clicks the AppIcon and the window reappears.

The release 2.0 Workbench has a special menu called ‘‘Tools’’. It is special because unlike the other Workbench menus, any application can add its own menu items to this menu. These menu items are called *AppMenuItems*. Like the AppIcon, the AppMenu can be used both as an activator and as a ‘‘drop box’’. When the user selects one of these menu items, Workbench sends a message to the application that created the AppMenuItem. If there were any icons selected when the user selected the AppMenuItem, the application will also get a list of those icons.

The AppMessage Structure

When Workbench notifies an application of AppWindow, AppIcon, or AppMenuItem activity, it sends an AppMessage to the application’s message port (from <*workbench/workbench.h*>):

```
#define      AM_VERSION     1

struct AppMessage {
    struct Message am_Message;    /* standard message structure */
    UWORD am_Type;                /* message type */
    ULONG am_UserData;            /* application specific */
    ULONG am_ID;                  /* application definable ID */
    LONG am_NumArgs;              /* # of elements in arglist */
    struct WBArg *am_ArgList;     /* the arguments themselves */
    UWORD am_Version;             /* will be AM_VERSION */
    UWORD am_Class;               /* message class */
    WORD am_MouseX;               /* mouse x position of event */
    WORD am_MouseY;               /* mouse y position of event */
    ULONG am_Seconds;             /* current system clock time */
    ULONG am_Micros;              /* current system clock time */
    ULONG am_Reserved[8];
};
```

The AppMessage’s am_Type field tells the application which type of AppObject the message is about. The field will be:

`MTYPE_APPWINDOW` if the message is about an AppWindow,
`MTYPE_APPICON` if the message is about an AppIcon, or
`MTYPE_APPMENUITEM` if the message is about an AppMenuItem.

When an application creates an AppObject, it can assign the AppObject application specific data (most likely a pointer) and an ID. Workbench will pass an AppObject's data and ID back to the application when it sends an AppMessage about the AppObject. The AppMessage's am_UserData and am_ID fields hold the user data and the ID.

The am_NumArgs field tells how many icons were involved in the user's AppObject action. For an AppWindow or AppIcon, am_NumArgs is the number of icons the user dropped on the AppWindow or AppIcon. For an AppMenuItem, am_NumArgs represents the number of icons that were selected when the user selected this AppMenuItem. If no icons were selected during an AppMenuItem event or the user double-clicked on an AppIcon, am_NumArgs will be zero. Workbench does not send AppMessages if the user double-clicks an AppWindow.

The am_ArgList field is a pointer to a list of WBArgs (from *<workbench/startup.h>*) corresponding to each icon dropped (or selected). If there were no icons dropped or selected, this field will be `NULL`.

For future expansion possibilities, the AppMessage structure has a version number. The version number is #defined as `AM_VERSION` in *<workbench/workbench.h>*.

The am_MouseX and am_MouseY fields apply only to AppWindows and contain the coordinates of the mouse pointer when the user dropped the icon(s). These coordinates are relative to the AppWindow's upper left corner.

The am_Seconds and am_Micros fields represent the time that the event took place.

Any remaining fields are undefined at present and should be set to `NULL`.

Adding AppObjects

The V37 *workbench.library* is made up of functions to add and remove AppObjects, two for each type of AppObject:

```
struct AppWindow *AddAppWindow( unsigned long myID, unsigned long userdata,struct Window
*mywindow, struct MsgPort *mymsgport, Tag tag1, ... );

struct AppIcon *AddAppIcon( unsigned long myID, unsigned long userdata, UBYTE *mytext, struct
MsgPort *mymsgport, struct FileLock *mylock, struct DiskObject *diskobj, Tag tag1, ... );

struct AppMenuItem *AddAppMenuItem( unsigned long myid, unsigned long userdata, UBYTE *menutext,
struct MsgPort *mymsgport, Tag tag1, ... );

BOOL RemoveAppWindow( struct AppWindow *appWindow );

BOOL RemoveAppIcon( struct AppIcon *appIcon );

BOOL RemoveAppMenuItem( struct AppMenuItem *appMenuItem );
```

The ‘‘AddApp’’ functions have several parameters in common. The myID and userdata parameters are values the application assigns to the AppObject. Workbench puts these values in the AppMessage’s am_ID and am_UserData fields when it sends an AppMessage about an AppObject. If an application receives AppMessages about several AppObjects at the same message port, the application can use the am_ID field to tell which AppObject Workbench is talking about.

The mymsgport field tells Workbench where to send this AppObject’s AppMessages. To make it easy to distinguish AppMessages from other types of messages, an application should devote a message port exclusively to AppMessages.

In the future, these AddApp functions will be able to process tag pairs in the parameter list. Currently, there are no tags defined for any of the AppObject functions.

All of the AddApp functions return a `NULL` if the function failed otherwise they return a pointer to a private structure. *The pointer serves only as a handle for the application to pass to the ‘‘RemoveApp’’ functions. Do not use it for anything else!*

Each of the RemoveApp functions removes one type of AppObject using the handle returned by the corresponding AddApp function. At present, these functions all return `TRUE`, but this behavior is not guaranteed to continue in the future.

AppWindows

The *workbench.library’s* AddAppWindow() call makes an application’s Intuition window into an AppWindow. It has one parameter that is different from the other AddApp calls, a window pointer. The mywindow field (from the prototype above) must point to an open Intuition window that is on the Workbench screen.

The C source code example *AppWindow.c* at the end of this article is a simple example of how to create an AppWindow.

There are two interesting things to note about the AppWindow. First, because an AppWindow is still an Intuition window, an application can use a Workbench AppWindow for any purpose it would need a normal Workbench based window for. An application can render graphics and text in it, process its IntuiMessages, or create menus for it. Also, because Workbench tells *where* on an AppWindow icons were dropped, an application can use a small region of a window as a drop box rather than the entire AppWindow. A program can even have several drop boxes on the same window. Using simple rendering routines, an application can draw the boxes so the user can see where to drop icons.

AppIcons

The *workbench.library* function AddAppIcon() adds an AppIcon to the Workbench window. There are three parameters unique to this AddApp function. The mytext parameter (from the prototype above) is the string that will appear beneath the AppIcon on the Workbench window. The diskobj parameter points to a DiskObject structure that Workbench will use for the AppIcon's imagery. It should be filled in as follows (from the *wb.doc* Autodoc):

```
diskobj - pointer to a DiskObject structure filled in as follows:
    do_Magic - NULL
    do_Version - NULL
    do_Gadget - a gadget structure filled in as follows:
        NextGadget - NULL
        LeftEdge - NULL
        TopEdge - NULL
        Width - width of icon hit-box
        Height - height of icon hit-box
        Flags - NULL or GADGHIMAGE
        Activation - NULL
        GadgetType - NULL
        GadgetRender - pointer to Image structure filled in as follows:
            LeftEdge - NULL
            TopEdge - NULL
            Width - width of image (must be <= Width of hit box)
            Height - height of image (must be <= Height of hit box)
            Depth - # of bit-planes in image
            ImageData - pointer to actual word aligned bits (CHIP MEM)
            PlanePick - Plane mask ((1 << depth) - 1)
            PlaneOnOff - 0
            NextImage - NULL
        SelectRender - pointer to alternate Image struct or NULL
        GadgetText - NULL
        MutualExclude - NULL
        SpecialInfo - NULL
        GadgetID - NULL
        UserData - NULL
    do_Type - NULL
    do_DefaultTool - NULL
    do_ToolTypes - NULL
    do_CurrentX - NO_ICON_POSITION (recommended)
    do_CurrentY - NO_ICON_POSITION (recommended)
    do_DrawerData - NULL
    do_ToolWindow - NULL
    do_StackSize - NULL
```

An easy way to create a DiskObject is to make an icon with the V2.0 icon editor, *IconEdit*. An application can call GetDiskObject() on the icon and pass that to AddAppIcon().

AddAppIcon()'s mylock parameter is for future enhancements and should be set to `NULL`.

Because AppIcons are Workbench icons, the user can drop them on an AppWindow or another AppIcon (or select them with an AppMenuItem). As there is no file, directory, or disk associated with an AppIcon (at least for the moment), the lock passed for the icon in `NULL`. Do not try to process icons with a `NULL` lock.

The C source code example *AppIcon.c* at the end of this article is a simple example of how to create an AppIcon.

AppMenuItems

Using the *workbench.library's* AddAppMenuItem() call, an application can add an AppMenuItem to the Workbench's ''Tools'' menu. This AppAdd function has one parameter unique to it, menutext (from the prototype above). It points to the string that appears in the ''Tools'' menu.

An AppMenuItem performs the same functions as an AppIcon or AppWindow, but it does not require the overhead of a DiskObject or a window. It also does not require the user to drop icons on an object. In some cases, the user might prefer to use an AppMenuItem over an AppIcon or AppWindow because the user doesn't have to shuffle around the Workbench windows to get to the ''Tools'' menu. Note that in older versions of release 2.0, Workbench did not supply a list of WBArgs when the user selected an AppMenuItem.

The C source code example *AppMenuItem.c* at the end of this article is a simple example of how to create an AppMenuItem.