AmigaMail

COLLABORATORS

	TITLE : AmigaMail	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 19, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

AmigaMail

1.1 IV-3: An Introduction to V36 Screens and Windows

by Ewout Walraven

> Editor's Note: The following article assumes the reader is familiar with the concept and implementation of tags in Intuition under release 2.0. More information on the subject can be found in the Amiga Mail article "TagItems and Tag Lists".

The new intuition.library provides the programmer with flexible yet simple methods of opening Intuition screens and windows. This article is an introduction to some of the Intuition and relevant graphics.library functions that make this possible. For more in depth information see the 1990 DevCon notes as well as the Autodoc and include files for release 2.0.

1.2 The Display Database

Release 2.0 of the OS introduces system support for different monitor types and programmable display modes. Applications can make use of these features through the Display Database. The Display Database provides an easy way for applications to stay compatible with past, present and future versions of the OS and the display hardware.

The database contains information about the available monitor types (like PAL or VGA) and the display modes that a particular monitor type is capable of displaying (like Lores, Hires, SuperHires, etc). This database makes it possible for the user to dynamically add new monitor types to the system, typically by calling the addmonitor command using a monitor spec file as an argument. These monitor types will be added to the database after the default monitor. The default monitor

properties are determined at boot time based on the configuration of the custom chip set. The default monitor is always either an alias for the NTSC or PAL monitor type.

When a monitor type is added to the database, the system figures out which display modes are available to this monitor type, based on the properties of the monitor's spec file and the configuration of the custom chip set. Later, the Display Database can be queried on the availability of a particular display mode using the appropriate graphics.library calls.

The following is an example of how to use graphics.library functions to find out what display modes are available. See the 2.0 Autodocs for a detailed description of the various functions.

```
    ULONG modeID;
    ULONG skipID;

    ULONG error, result;
    struct DisplayInfoHandle displayhandle;
    struct DisplayInfo displayinfo;
    struct NameInfo nameinfo;

    /* graphics.library must be opened */

    /* Don't want duplicate entries in the list for the
     * 'default monitor', so we'll skip the the videomode
     * for which default.monitor is the alias.
     */

    /* INVALID_ID indicates both the start and the end of the list of
       available keys */
    modeID = INVALID_ID;

    GetDisplayInfoData(NULL, (UBYTE *) & displayinfo,
            sizeof(struct DisplayInfo), DTAG_DISP, LORES_KEY);
    if (displayinfo.PropertyFlags & DIPF_IS_PAL)
        skipID = PAL_MONITOR_ID;
    else
        skipID = NTSC_MONITOR_ID;


    /* Given a ModeID, NextDisplayInfo
                        returns the next ModeID in the list */
    while((modeID = NextDisplayInfo(modeID)) != INVALID_ID) {

        /* Skip it? */
        if (modeID & MONITOR_ID_MASK != skipID) {

            /* ModeNotAvailable returns NULL if a displaymode,
               specified by the modeID, is available, or an error
               indicating why it is not available.
            */
            if ((error = ModeNotAvailable(modeID)) == NULL) {

                /* This displaymode is available, get the naming
```

```
                information. GetDisplayInfoData, can either be
                called with a handle to a displaymode record, or
                the display modeID. Never use the handle directly.
            */

            /* returns NULL if not found */
            if (displayhandle = FindDisplayInfo(modeID)) {
                result = GetDisplayInfoData(displayhandle,
                        (UBYTE *)&nameinfo, sizeof(struct NameInfo),
                        DTAG_NAME, NULL);
                if (result)     /* 'result' indicates the number
                                   of bytes placed in the buffer */
                    printf("%s is available.\n", nameinfo.Name);
            }
        }
    }
}
```

Using the GetDisplayInfoData() function, this example asks only for the descriptive name of the display mode via the NameInfo structure. Using this same function with a different tag, other information on a particular display mode can be obtained. GetDisplayInfoData() can supply information about the properties of the display mode (DisplayInfo), the display mode's dimensions (DimensionInfo) or the display mode's monitor specifications (MonitorInfo). Refer to graphics/displayinfo.h for a description of the various structures.

1.3 Opening Screens and Windows

Intuition V36 supplies two new functions to open screens and windows: OpenScreenTagList() and OpenWindowTagList(). They provide added functionality over the older OpenScreen() and OpenWindow() functions. Instead of the NewScreen and NewWindow structures, an array of parameters called tags is passed to the function in the form of a TagItem structure pointer. This array is known as a tag list. For a description of using tags in Intuition, see the Amiga Mail article ''TagItems and Tag Lists''.

The following is a simple example of the usage of OpenScreenTagList() and OpenWindowTagList(), using only a tag list as a parameter to each function.

```
struct TagItem tagitem[3];
struct Screen *screen;
struct Window *window;

tagitem[0].ti_Tag  = SA_DisplayID;
tagitem[0].ti_Data = HIRES_KEY;     /* Use HIRES displaymode
                                       for default monitor */
tagitem[1].ti_Tag  = SA_Title;
tagitem[1].ti_Data = "AmigaMail Test Screen";
tagitem[2].ti_Tag  = TAG_DONE;    /* Marks the end of the tag array. */
```

```
    /* All others options will be set to default. Note this screen will not
       open with the 'New Look', due to the absence of the SA_Pens tag and
       data. Further examples will have this tag */

    if (screen = OpenScreenTagList(NULL,   tagitem)) {
        tagitem[0].ti_Tag = WA_CustomScreen;
        tagitem[0].ti_Data = screen;    /* Open on my own screen */
        tagitem[1].ti_Tag = WA_Title;
        tagitem[1].ti_Data = "AmigaMail Test Window";
        tagitem[2].ti_Tag = TAG_DONE; /* Marks the end of the tag array. */

        /* Use defaults for everything else. Will open as big as the screen. */

        if (window = OpenWindowTagList(NULL, tagitem)) {

            /* rest of code */

            CloseWindow(window);
        }
        CloseScreen(screen)
    }
```

See the intuition/intuition.h and intuition/screens.h include files for a complete list of currently supported tags.

Intuition.library provides a function to get information about screens and display modes called GetScreenDrawInfo(). This function returns a subset of the data supplied by the graphics.library's GetDisplayInfoData() function. Given a pointer to a screen, GetScreenDrawInfo() returns a pointer to a DrawInfo structure describing that screen. Note that this structure is read-only. The DrawInfo structure contains information about the resolution, default font and pen colors of the screen. Since the system allocates the DrawInfo structure when GetScreenDrawInfo() is called, a program must tell the system to free it by calling FreeScreenDrawInfo(). Of course, a program should only release the DrawInfo structure after it is no longer needed.

When using GetScreenDrawInfo() on a public screen, the screen should first be locked using Intuition's LockPubScreen() function. This will prevent a public screen from closing while your program is looking at the screen's DrawInfo structure. UnlockPubScreen() will unlock the screen. Public screens are discussed in more detail later in this article.

The graphics.library function GetVPModeID() provides a way of determining the display mode ID associated with a ViewPort. Passed the ViewPort attached to a screen, the function will return the display mode ID for that ViewPort, or INVALID_ID if no display mode ID is associated with the ViewPort.

An example of the usage of GetVPModeID() and the other functions discussed so far in this article are provided in at the end of thisarticle in example 1, CloneWB.c. This example also uses the stack-based amiga.lib support library functions OpenScreenTags() and OpenWindowTags(), both of which are documented in the release 2.0

Autodocs.

1.4 Public Screens

V36 introduces the public screen, a custom screen on which other applications can open their windows. A familiar example of a public screen is the Workbench screen. The public screen provides the ability to create system supported environment alternatives to the Workbench.

Opening a public screen is just like opening a custom screen. The only difference is in the tags used when the screen is opened. There must be a SA_PubName tag, with the screen's name as the tag's data. Applications will access the screen by this name. When a public screen is first opened, it is still private to the application that opened it. This will give the program that opened the screen the opportunity to set up before other applications can access the screen.

After a public screen is explicitly made public, anybody can put a lock on the screen or open a window on it. Both locks and 'visitor windows' prevent applications from closing their public screens. Intuition keeps track of the number of locks and visitor windows on a public screen by incrementing and decrementing the screen's visitor count. When the number of visitors is zero, the screen can be closed.

The following code fragment shows how to open a public screen and change its status from private to public using the Intuition function PubScreenStatus().

```
    if (pubscreen = OpenScreenTags(NULL, SA_DisplayID, HIRES_KEY,
                    SA_Depth, 2, SA_Title, "AmigaMail Test Public Screen",
                    SA_PubName, "AmigaMail Test Public Screen",
                    /* By providing the public name, you indicate
                       it will be public */
                    SA_Pens, dri_Pens,
          SA_ErrorCode, &oserror,
          TAG_END)) {
        /* change mode from 'private' to 'public' */
        oldstatus = PubScreenStatus(pubscreen, 0);}
```

If for any reason Intuition could not open the screen, the error number will be returned in the data portion of the SA_ErrorCode tag (the OpenScreen errors are listed in intuition/screens.h). This error can indicate that the system is out of memory, that the requested display mode is not supported by the video hardware or monitor, or that the public screen name is already being used by another public screen.

Some OpenScreen errors can be avoided. Make sure the display mode a screen requires is available (one of the previous examples shows how to find all available display modes). For example, before opening a public screen, make sure its name is not already being used by an existing public screen. Before opening a public screen named

ExamplePublicScreen, try to put a lock on a screen with that name using LockPubScreen(). If this call succeeds, the system has found and locked a screen with that name, so you can't use it. If the call fails, the system does not have a screen by that name, indicating that it's OK to open the new screen with that name. An application should make sure it removes all locks on a screen when the screen is no longer needed.

If a screen already exists with that name, there are several options available, all of which should reflect user preferences. One option is not to open a screen, using the existing screen instead. For example, if the user double-clicks an application's project icon, the application would check if its public screen is already opened and open windows there before trying to open a new screen. Another option is to open a new screen with a slightly different name. A public screen that would have been named ExamplePublicScreen might instead be called ExamplePublicScreen2. A third option is to quit. In any case, the user should normally be choosing the actions of the application, not the programmer.

Programs find specific public screens by their name, so public screens must have a unique name as identification. With a maximum name length of 139 characters, supplying a unique name shouldn't be a problem. This name should reflect the nature of the screen's purpose, if for no other reason, to make it easy to figure out what the screen is used for. Ambiguous names such as "PublicScreen" or "MyScreen" should not be used as they tell nothing about the nature of the screen. Also, more than one "less creative" programmer might be inclined to give such generic names to their public screen, which can confuse both user and application.

As a public screen can be used as an alternative to the Workbench environment, programs opening public screens should normally make the screen is at least 640 pixels wide. Many applications assume that the Workbench is at least 640x200 and would be severely crippled if opened on a 320x200 screen.

Before a public screen can be closed, it has to be made private again. As long as there are visitors (windows and locks) on a public screen, an attempt to change the screen's status to private will fail. To make closing down public screens easier, Intuition can signal an application when the last visitor 'leaves' a public screen. While waiting to close a locked public screen, an application can start to shut down by unloading code or closing windows. After the screen has been made private again, Intuition removes it from the public screen list. The following code illustrates this.

```
if ((oldstatus = PubScreenStatus(pubscreen, PSNF_PRIVATE)) & PSNF_PRIVATE)
            CloseScreen(pubscreen);
```

To tell Intuition to signal when all the visitor windows on a public screen have been closed, use the SA_PubSig and SA_PubTask tags when opening the screen. The SA_PubSig data is the number of a signal bit you have allocated. An application can specify which task to signal when all the visitors have left by supplying a pointer to that task

with the SA_PubTask tag. If this tag is absent or its data = NULL, the task which opened the public screen will be signaled.

When putting up any requesters on a public screen, be sure to supply the requester with a pointer to a visitor window on that screen. This will prevent the requester from popping up on the wrong screen. This will also prevent needless incrementing of the public screen's visitor count as the requester will not be considered a visitor.

Prior to release 2.02, UnlockPublicScreen() did not signal when the visitor count reached zero. With release 2.02, UnlockPubScreen() will signal the specified task when the visitor count reaches zero. Under 2.00 and 2.01, only CloseWindow() would signal.

There is always a default public screen available. Normally this would be the Workbench, but an application can make its own public screen the default, or better yet, a screen manager utility could let the user choose the default public screen. An application should not automatically make its public screen the default as this might confuse the user and can conflict with user preferences.

Because older programs do not know about public screens, The SHANGHAI flag has been provided to 'shanghai' requests to open a window on the Workbench screen, forcing the window to open on the default public screen. Of course, a release 2.0 application that wants to open a window on a specific public screen can do so provided that the specific screen is available. Another flag, POPPUBSCREEN, instructs Intuition to move public screens to the front if a window is opened on it. The SHANGHAI and POPPUBSCREEN flags can be set using PubScreenStatus() but, as the flags are global and should be set according to the preferences of the user, applications should not be changing them.

Example 2, Pub.c, shows how to open a public screen, make it the default and shut it down. The example can be found at the end of this article.

To aid 'screen manager' utilities, the list of public screens can be locked and copied using LockPubScreenList(). UnlockPubScreenList() will unlock the list. Don't try to interpret the list itself. Instead, copy it and release the lock. To receive an updated copy later on, lock and copy the list again. Lockpub.c is a small example of locking the public screen list.

1.5 Visitor Windows

Opening a visitor window on a public screen is like opening a window on a custom screen except that the window is specifically defined as a visitor. This is done when opening the window by either replacing the CUSTOMSCREEN flag in the (Ext)NewWindow->Type field with PUBLICSCREEN or using a WA_PubScreenName or WA_PubScreen tag. To use the PUBLICSCREEN flag or the WA_PubScreen tag (this tag has a pointer to the public screen as data), the screen must be prevented from closing. Normally this is done by locking the desired public screen. In either case the program must provide a pointer to the public screen if it

requires a specific public screen. If an application has no preference towards a particular screen, it can pass NULL as the data for the WA_PubScreen tag to open a window on the default public screen.

The following example shows how to open a visitor window on a specific public screen or, if that screen can't be found, on the default public screen.

```
    struct Screen *screen;
    struct Window *window;
    UBYTE namebuffer[MAXPUBSCREENNAME];

    if (!(screen = LockPubScreen("CATScreen"))) {
        /* Can't lock CATScreen, fall back on default */
        /* The method used in this example to fall back on the default
           screen is not necessary and is provided only to illustrate
           the usage of GetDefaultPubScreen().
        */
        GetDefaultPubScreen(namebuffer);
        /* Lock it, so it can't go away. */
        /* If in this split second another screen is made the default
           and this one closes, screen would become NULL and the window
           will open on the default public screen anyway.
        */
        screen = LockPubScreen(namebuffer);
    }
    if (window = OpenWindowTags(NULL,
                    WA_Left, 0, WA_Width, screen->Width,
                    WA_Top, screen->BarHeight,
                    WA_Height, screen->Height - screen->BarHeight,
                    WA_PubScreen, screen,
                    WA_Flags,
                    WINDOWDRAG|WINDOWDEPTH|WINDOWCLOSE|ACTIVATE|
                              SIMPLE_REFRESH|NOCAREREFRESH,
                    WA_Title, "AmigaMail Visitor window", TAG_END)) {
        /* Don't need the lock anymore. */
        UnlockPubScreen(NULL, screen);
                        .
                        .
                        .
        CloseWindow(window);
    } else UnlockPubScreen(NULL, screen);   /* OpenWindow failed for
                                            some other reason */
```

In this code example, once it is locked (first by LockPubScreen(), later by the visitor window) the screen can't go away until all its visitors have left. After opening its initial visitor window, an application can open non-visitor windows on a public screen, however these windows will have to be closed before releasing the lock on the public screen.

When opening a visitor window, the public screen on which it opens can be specified by using a pointer to the screen (with the WA_PubScreen tag) or by supplying the screen name (using the

WA_PubScreenName tag). This latter method is particularly useful where an application prefers a specific public screen, but can run using the default public screen. If the preferred public screen is not present when an application attempts to open a visitor window, the window will either open on the default public screen or will fail to open depending on the WA_PubScreenFallBack tag. If this tag is present and is set to TRUE, the window will open on the default screen. If WA_PubScreenFallBack is not present or is set to FALSE, the window will not open.

1.6 Overscan

In release 2.0, Intuition fully supports video overscan. There are four standard overscan specifiers: OSCAN_TEXT, OSCAN_STANDARD, OSCAN_MAX and OSCAN_VIDEO. OSCAN_TEXT is based on user preference settings and indicates a display which is within the visible bounds of the monitor. The dimensions of this type of overscan are used for the Workbench screen and for the defaults for OpenScreenTagList(), STDSCREENWIDTH and STDSCREENHEIGHT. OVERSCAN_STANDARD is also based on user preference settings. The edges of an OVERSCAN_STANDARD display are set by the user via Preferences to be just outside the visible bounds of the monitor. OSCAN_MAX is the largest display fully supported by Intuition. OSCAN_VIDEO is the largest display region which, for a given display mode, can be reliably generated. Note that OSCAN_VIDEO displays are not scrollable.

The Intuition function QueryOverscan() returns rectangle information for the indicated overscan type of a particular display mode. Using the rectangle returned by QueryOverscan(), it is possible to explicitly specify the offset and dimensions of the DisplayClip region. Note however that this custom rectangle must fit within the OSCAN_MAX rectangle, offset included. It is not permitted to specify custom rectangles whose values are in between OSCAN_MAX and OSCAN_VIDEO, nor is it permitted to specify rectangles larger than OSCAN_VIDEO.

1.7 DisplayClip

The DisplayClip defines the on-monitor region, and is independent of the screen dimension. The rectangle describing the DisplayClip determines how much of the screen can be seen at a time. The screen is "shown through" the DisplayClip in much the same way a SuperBitmap is shown through a window. If the screen is larger than the DisplayClip, only part of the screen will be visible. The DisplayClip makes it is possible for Intuition to support hardware scrolling of a raster larger than can fit on the display. Intuition will take care of specifying the rectangle of the DisplayClip region based on the default screen dimensions or based on the SA_Overscan tag (if it is present), so most programs shouldn't have to specify the DisplayClip rectangle.

Example 3, ScreenDisplayModes.c, illustrates opening any type of

screen currently available to the system, using any type of available
overscan.