Q&A

Q: Is it possible to have one IDCMP *port for two windows on two screens?*

A: Yes. The standard technique for sharing one IDCMP port is:

```
sharedport = CreatePort( ... );
        ...
newwin.IDCMPFlags = 0;
        ...
win = OpenWindow( &newwin );
win->UserPort = sharedport;
ModifyIDCMP( win, DESIRED_IDCMPFLAGS );
        ...
/* Open other windows the same way */
        ...
/* You must use CloseWindowSafely() to close
windows with shared IDCMP ports -- CloseWindow()
alone is unsafe... */
CloseWindowSafely( win );
```

Notice that this code uses the function CloseWindowSafely() (see the CloseWindow() *Autodoc* for the code). This function is specially designed to clean up after a window that shares an IDCMP port either with a window or some other inter-process communications customer. *Don't modify the loop in the CloseWindow() Autodoc! It is correct.* Some people think they see a bug there (one iteration too many), but they're wrong.

Q: How do I provide a Zorro II 24-bit DMA device with a few bytes of memory on any machine?

A: Call the AllocMem() function using the new (for 2.0) MEMF_24BITDMA flag defined in <exec/memory.h>. (It is much like MEMF_CHIP and MEMF_FAST only for 24BITDMA...)

Q: but what if I'm using a 1.3 system?

A: If you call AllocMem() with the MEMF_24BITDMA flag, the pre-2.0 AllocMem() will still check for that flag. You could write a utility that, under pre-2.0 systems, adds the MEMF_24BITDMA flag to a memory header's MH_ATTRIBUTES field of the 24-bit memory areas.

Q: With some programs I get a lot of enforcer *hits (write-long) at addresses $180, $184, $188, etc. with PC in ROM at location $F830C. Why?*

A: See the *Autodoc* for *exec.library/Alert*. This is where exec stores the processor registers when there is an alert caused by a processor trap. You should be able to make some guesses about what's wrong with your program by looking at the register values being written. The Guru Number is in D7.

Q: Commodore has publicly stated that applications should not *make specific use of the PMMU. Is this correct?*

A: Yes. While tools/hacks sometimes use the PMMU (such as *Enforcer*) they can not be expected to work when the system itself starts to use the PMMU. The PMMU is a supervisor-space feature of the CPU and, for it to do what it does, it must remain a system controlled resource. If you wish to do MMU-related things in an OS and processor compatible manner, use the function calls available in 2.04 (such as CachePreDMA(), CacheControlU(), etc). These will do their best to ‘‘do the right thing’’ on all OS/CPU combinations.

More complex usages (private MMU tables, etc) are the domain of the OS and of various hacks (but often highly useful hacks) that may not work with future OS/CPU combinations.

Q: When opening a `BORDERLESS` *window under 2.0, do you have to also make it* `BACKDROP` *to stop it from having Borders?*

A: A window has borders unless *all* of the following are true:

1. It is declared `BORDERLESS.`
2. It has no title (NewWindow.Title = `NULL`, not "").
3. It has no user gadgets in the border (i.e. gadgets with `xxxBORDER` set).
4. It has no system gadgets in the border (i.e. no close gadget, no size gadget, no depth gadgets, etc.).

Now the difference with 2.0 is that the NewLook borders are rendered when a window goes active or inactive. Under 1.3, the borders are not re-rendered when the window becomes inactivate.

Some programs have a close gadget, and when they open their window, they write over their border. They may also have to overwrite it when they get an `ACTIVEWINDOW IDCMP` message. Under 2.0, this trick/kludge isn't enough.

`BACKDROP` does not enter into the picture.

The real trouble is that Intuition does not directly support putting gadgets like the close gadget into borderless windows. It never did, and the kludge that makes it basically work under 1.3 is not sufficient under 2.0.

In general, It's not a good idea, stylistically, to have borderless windows that are not backdrop windows, since that may make it difficult for the user to see the window bounds against other windows behind it.

Q: Some of the format specifiers for the dos.library function VFWriteF() contain a numerical value for field lengths. The documentation states that the numerical value is in base 36! Why is base 36 used?

A: It is a remnant from BCPL. It is the base you get when using all of the alphanumeric characters.

Q: I am trying to get the number of display rows from the Workbench screen. What is the proper procedure for obtaining the number of Rows?

A: Under 2.0, you should use the Graphics Display Database to get this information.

```
struct DimensionInfo MyDimInfo;
struct Screen *WBenchScreen;
WBenchScreen = LockPubScreen("Workbench");
id=GetVPModeID(&(WBenchScreen->ViewPort));
GetDisplayInfoData(NULL,&MyDimInfo,
        sizeof(MyDimInfo), TAG_DIMS, id);
```

MyDimInfo will now contain the sizes and positions for Nominal, Video Overscan, Text Overscan, and Standard overscan. Text and Standard are from the user's preferences settings.

Q:How do I rename a disk?

A: Under 1.3, you have to send the `Action_Rename_Disk` packet to the disk's filesystem. See the *AmigaDOS Manual* for more details on this packet. To find out where to send the packet, use DeviceProc()on the old name (plus the ‘‘:’’) or the device name (i.e.: ‘‘df0:’’).

Under 2.0, you can use the new Relabel() function of *dos.library*. That function basically sends an `Action_Rename_Disk` packet.