For the first PCI systems, the "new driver model" will only be required for PCI hardware
interface drivers. Existing 68000 drivers will work without change. As we discussed at WWDC, the new model will be used for all device drivers when the fully-native operating system is available. The new device-driver model is currently for hardware-interface drivers only and it is not clear at this time how virtual drivers (such as an AppleTalk ATP driver that calls an EtherNet driver) will fit into this model.

To write drivers that can be easily transported to the new model, you need to do a few things.
This is a summary, and we will make more information available when the design stabalizes:

1. Write your drivers in very clean C or C++. Except for a framework that interfaces to the Device Manager calls, eliminate all assembler from your driver.

2. Do not call the Toolbox Managers: very few will be supported for drivers. You should be able to call the following:

-- Deferred Task Manager: do as much as possible using Deferred Tasks. The Deferred Task Manager will be replaced by a secondary-interrupt service.

-- Device Manager:
  -- Write a framework that extracts relevant information from the Device Manager call (asynchronous, immediate, read vs. write) and calls a separate function to handle the request.
  -- Don't access or modify the DCE.
  -- Don't fake calls to IODone. Immediate calls must return their result in their return to the Device Manager. non-immediate calls must call IODone with correct final status. Never jump to JIODone. Rather call the IODone routine and return to the Device Manager. Make sure your driver is fully re-entrant, as it may be re-called after calling IODone.
  -- Drivers will receive initialize and terminate calls as their first and last calls from the Device Manager. Think about how you will support this in your driver, For example, you can call your initalizer when OpenDriver is first called, for example.
  -- Open and Close will become connection-oriented. Currently, the first Open means "Initialize" and whether "Close" terminates is undefined.
  -- Respond to all Device Manager calls. The "readEnable," "writeEnable," etc. bits in the DCE will be ignored. (Of course, you return an

error if you don't support a call.)
-- Don't use the dCtlEMask and dCtlMenu fields. Device Drivers are not Desk Accessories.
-- Concurrent drivers will be supported. If your drivers process multiple simultaneous operations, isolate the code that manages the driver queue as it will have to be replaced.
-- In this model, drivers are the interface between the Mac O.S. and some piece of hardware. They are at the bottom-end of the food chain and should not make PBRead (etc.) calls to other drivers. This may change when the new driver model is extended to the fully-native operating system.

-- Gestalt Manager calls are supported. However, drivers should implement the new Driver Gestalt calls.

-- Avoid the Memory Manager. Call NewPtrSys and DisposePtr from your initialize and terminate routines (i.e, from Open and Close). Don't call them elsewhere. A new driver-specific memory management service will be provided.

BlockMove (BlockMoveData) is supported.

-- Avoid the Notification Manager. I'm not certain at this time whether it will be supported for drivers.

-- The following O.S. Utilities will be supported:
-- Debugger, DebugStr
-- Enqueue, Dequeue

-- Resource Manager calls will not be supported, not even in Open and Initialize calls. There will be a "System Registry" capability that serves the purpose of configuration so, if you must call the Resource Manager, do it in an Init, Control Panel or (last choice) in an intentionally non-transportable part of your Open routine.

-- Stay away from the Segment Loader.

-- Time Manager calls will be provided for absolute- and up-time. Time-based scheduling (wake-up events triggered by Time Manager completion routines) may be supported, but this is not yet clear. Avoid the Vertical Retrace Manager traps (I'm not sure how/if they will be supported).

3. Our intent is that PCI drivers written to the new standard will be transportable

without change to the fully-native operating system. This is one motivation for the extremely limited access to the Toolbox in this model.

Please note that the above is a summary of work-in-progress. The design should stabilize in a month or so and we have every intent of making this information widely available as soon as possible so that driver writers can plan their conversion effort. Right now, the best thing you can do is to write your code in very clean and straight-forward C and isolate initialization, interface, and configuration from your task-specific code.