The 'cdev' Resource

In order to function, a cdev must have an interface that the **Control Panel** understands and can work with. This interface consists of the resources necessary for the **Control Panel** to open and display the cdev, and the code that executes the control panel device's function, including responses to queries from the **Control Panel**.

A cdev must have specific resources of the types: 'DITL', 'mach' , 'nrct', 'ICN#', 'FREF', 'BNDL' and 'cdev'; they should have resources ID of -4064. All of the cdev's resources must fall in the resource ID range between -4033 to -4064 and are further divided according to whether they're to be used by the **Control Panel**'s half of the interface, or the cdev's half. ID's from -4064 through-4049 belong to the **Control Panel** while those of -4048 to -4033 belong to the cdevs.

The 'BNDL', 'ICN#' and 'FREF' resources tell the Finder and the **Control Panel** how to display the cdev. Additionally, the Finder needs to see an owner resource to display the cdev's icon. See Finder Icons & BNDLs for more information.

'DITL' describes the cdev's dialog item list, which the **Control Panel** adds to its own when a user activates a cdev. Since a cdev's dialog items must fit within the overall Control Panel window, the coordinates for its 'DITL' must place it inside a rectangle described by the points 1, 89, 253, 320.

Not all cdevs are able to run on all Macintoshes. That being the case, the 'mach' or "machine" resource tells the **Control Panel** if there is a match between the requirements of the cdev and the capabilities of the machine. The first two bytes of the 'mach' resource is the "Softmask" which is compared against the low-memory global ROM85 for available Toolbox features. The second two bytes is the "Hardmask" which is compared with the low-memory global HwCfgFlags for hardware match-ups.

A cdev will only appear if every 0 bit in the Softmask is matched by a corresponding 0 bit in ROM85 and if every Hardmask 1 bit is matched up with a HwCfgFlags 1 bit. To indicate that a cdev can run on all Macintoshes, its Softmask should be set to 0xFFFF and the Hardmask to 0x0000 . Also, a 0x0000 in Softmask and a 0xFFFF in Hardmask causes the **Control Panel** to send a macDev message to the cdev. At this point, the cdev should test to see if it's able to run (by using **SysEnvirons** or **Gestalt**, for example), and it should return 1 if it can run, and 0 it if can't.

Resource type 'nrct' describes a rectangle list, beginning with the number of rectangles in the list (two bytes wide) and followed by 8-byte descriptions of each rectangle-giving its location in top, left, bottom, right order. 'nrct' and 'DITL' coordinate the appearance of the control panel. The overall size of the 'nrct' is two pixels smaller in every direction than the 'DITL', so rectangles must be placed inside the coordinates -1, 87, 255, 322. The **Control Panel** fills in all areas that aren't covered by rectangles with ltGray shading.

The 'cdev' resource contains the code that the **Control Panel** uses to send cdev messages to. See cdev for a description of how to write a cdev code resource.

Messages

The message field of the cdev function carries checking, status, management, and functional information between the **Control Panel** and a cdev. A total of 15 specific messages, each indicated by a different value, let the **Control Panel** tell a cdev what just happened.

initDev is identified by the value 0 in the cdev function's message field. The **Control Panel** sends this message to a cdev when the user first clicks on the cdev's icon. It lets the cdev obtain any memory it might need and sets buttons and controls to their initial settings. If a cdev needs storage that will be allocated as long as it's open, it should allocate this storage using **NewHandle**, and it should return this handle as a result of this message. If a cdev doesn't need memory, then it should return the value that it received in the *cdevValue* parameter to thecdev function.

hitDev is identified by the value 1 in the cdev function's message field. The **Control Panel** sends a hitDev message whenever the user clicks on one of the cdev's enabled dialog items. That item's number is passed in the cdev function's Item field.

Since the **Control Panel**'s dialog items occur before the cdev's items in the **Control Panel**'s dialog item list, the Item field may not be the same as the item number in the cdev's 'DITL'. The cdev function parameter numItems contains the number of items that the **Control Panel** has in its dialog item list, so the cdev function should subtract numItems from Item before checking to see which dialog item was hit. If the cdev needs to call **Dialog Manager**, it will have to add numItems to the 'DITL' item number first. One technique is to write wrapper functions for the **Dialog Manager** routines that require item number arguments, and simply add numItems locally for those routines. Don't assume that numItems is a constant, since the number of dialog items in the **Control Panel** may change in future System Software releases.

closeDev is identified by value 2 in the cdev function's message field. The **Control Panel** sends the cdev a closeDev when the user selects another cdev or closes the **Control Panel**. If the cdev function allocated storage on the initDev message using the *cdevValue* field, then it should dispose of that memory at this point. Note that the dialog items of the cdev have already been disposed of by the **Control Panel** at this point, so a cdev can't try to examine the value of a dialog item. If a cdev has an editText field, for example, then it should keep track of the characters entered into this field as it's typed. Otherwise, the contents of this field will be lost.

nulDev is identified by value 3 in the cdev function's message field. This message is sent to the cdev so it can perform a periodic action, such as flashing a cursor (note that the insertion points for active editText items are already being blinked by **DialogSelect**). Since the cdev can't depend on how often you'll receive nulDev messages, you shouldn't use it to perform time-dependent tasks

updateDev is identified by value 4 in the cdev function's message field. The **Control Panel** sends the cdev an updateDev whenever the contents of the cdev part of the **Control Panel** needs to be redrawn. This message is analagous to the **Window Manager** event updateEvt. The cdev should take care of any updating that isn't already handled by the **Dialog Manager**. Because the update region is reset before **Control Panel** sends the updateDev message, your cdev must completely redraw all of your user items.

activDev is identified by value 5 in the cdev function's message field. The **Control Panel** sends the cdev an activDev message every time the **Control Panel** is activated. The cdev can then respond by reactivating any items that the cdev disabled when it received a deActivDev message.

deActivDev is identified by value 6 in the cdev function's message field. Whenever the **Control Panel** is deactivated, it sends a deActivDev message to the cdev. The cdev should properly disable its items in the **Control Panel**. If the cdev has any editText items, they should be disabled. If the cdev displays a selection in a scrolling list, or has a selection of some other kind, it should be disabled as well.

keyEvtDev is identified by value 7 in the cdev function's message field. The **Control Panel** sends the cdev a keyEvtDev message on every keyDown and every autoKey event, thus letting the cdev process those events. When the key event is then passed back to the **Control Panel**, the **Dialog Manager**'s **DialogSelect** function processes the result. Change the what field in the cdev function argument theEvent to nullEvent before returning to the **Control Panel** if you don't want **DialogSelect** to process the key event.

macDev is identified by value 8 in the cdev function's message field. This is the first message a cdev gets whenever the 'mach' resource has 0x0000 in Softmask and 0xFFFF in Hardmask. It lets the cdev determine if it can run on this individual Macintosh. The cdev responds with a 0 if it decides that the **Control Panel** should not display its icon and with a 1 if it decides that this machine has the proper requirements to run it.

undoDev, cutDev, copyDev, pasteDev, and clearDev are identified by values 9, 10, 11, 12 and 13, respectively, in the cdev function's message field. They are standard edit menu messages and are sent by the **Control Panel** to the cdev whenever the user selects one of these functions from the Edit menu. Note that command-key equivalents for these operations are *not* handled by the **Control** Panel, and are sent to the cdev with the keyEvtDev message.

cursorDev is identified by value 14 in the cdev function's message field. The **Control Panel** sends a cursorDev message as long as the cdev contains a 'CURS' resource with an ID below -4064. It lets the cdev define the shape of the cursor to something other than the standard cross whenever the cursor is within the cdev-controlled part of the Control Panel window.

It important to remember that you have to draw your user items in response to updateDev messages rather than using a draw item function (see **SetDItem** for more infomation). Since the cdev's code may be unloaded or moved between calls, it's likely that the draw procedure pointer will become invalid. Also, if your cdev needs to reference QuickDraw globals, it will have to reference them directly off of register A5, using CurrentA5.

Error Handling

Errors in cdevs usually result from either not enough memory or from missing resources. The **Control Panel** can help you report these common errors, but the cdev itself must be able to respond to more specific problems. Usually, your cdev should be equipped to handle errors on its own. If your cdev does encounter an error, it should release all memory that it allocated, and return an error code.

When a cdev wants to report an error, it can send a message to the **Control Panel**, or put up its own alert.1) The cdev can ask the **Control Panel** to display an out-of-memory alert and gray out the **Control Panel**'s cdev-controlled area. 2) The cdev can ask the **Control Panel** to display an alert telling the user it can't find a necessary resource, and also gray out the **Control Panel**'s cdev area. 3) The cdev can display an alert that it provides and then ask the **Control Panel** to gray out the area it controls.

The *cdevValue* field can be used to report errors to the cdev. Under normal circumstances *cdevValue* passes the same value back and forth: the last function value (potentially including memory allocation) the **Control Panel** received from the cdev fills the field when the **Control Panel** calls cdev and the same number goes back to the **Control Panel** again in the return.

If everything goes fine, *cdevValue* will be a constant defined as cdevUnset. When something goes wrong, however, one of three other constants replaces the cdevUnset function value in the *cdevValue* field.

A return value of cdevGenErr means that an error of an unspecified, or generic, type has occurred. No error dialog is displayed but the **Control Panel** dims the cdev's part of the window and puts a 0 in the *cdevValue* field for calls thereafter.

A return value of cdevMemErr means that the cdev didn't find enough memory to execute. A memory error dialog is displayed and the Control panel dims the cdev's part of the Control Panel window and sends 0 in the *cdevValue* field for subsequent calls.

A return value of cdevResErr means that a required resource couldn't be found. A resource error dialog is displayed and the **Control Panel** dims the cdev's part of the Control Panel window and sends a 0 in the *cdevValue* field for subsequent calls.

The **Control Panel** will ignore all return values from a cdev after it responds to an error message.

A possible source of problems that involves neither memory nor resources concerns cdevs whose 'DITL' contain editText items. When cdevs include editText items, the **Dialog Manager** allocates a TEHandle to use for text editing and display. Only then does the **Control Panel** send an initDev message.

The problem arises when an error is encountered after the TEHandle has been allocated, because it is not disposed of when a cdev goes through its shutdown procedure. The **Dialog Manager** sees the TEHandle and continues to flash the insertion point. When the user attempts to enter text, the **Dialog Manager** tries to process the key-down events, and the system crashes.

To avoid causing system crashes with cdevs that have editText items, the cdev should hide its editText items with **HideDItem**. As long as an item is invisible, the **Dialog Manager** will not try to process keyDown events, and therefore will not cause an error.