Chapter 4: Mass Storage Control Interface

The Mass Storage Control Interface routines provide a means for the Operating System to ask the driver to examine the devices attached to a specific adapter, and to register any devices not previously registered with the OS. The routines are entered at blocking process level and may call any blocking or non-blocking routines. It is suggested that these routines call *CYieldIfNeeded* (v4.xx), *CRescheduleLast* (v3.1x), or *DelayMyself* if it is necessary to spend any significant time waiting for completion of required functions (such as a scan of the SCSI bus for new devices). These routine entry points are supplied by the driver in the *AddDiskSystem* call, and thus may be different for different adapter cards.

ScanForDevices

Disk and other device drivers must provide a procedure which the Operating System can call to cause the driver to scan for new un-registered mass storage devices, and to register them with the OS. The operating system passes the *CardStructure* pointer on the stack as the only parameter when it makes the call to the drivers *ScanForDevices* procedure for each adapter card. The syntax for the OS call to the driver *ScanForDevices* entry point is as follows:

```
void ScanForDevices(
        CardStruct *CardHandle);
```

where: CardHandle Card structure pointer (handle) returned by *AddDiskSystem* call

Once the device has been registered with the OS, the driver must be prepared to handle normal I/O requests and activity.

The driver's *ScanForDevices* procedure is called from the OS at blocking process level. This allows the procedure to make blocking calls such as *AddDiskDevice*. The driver should scan to determine if there are any devices attached to the adapter card which have not been registered with the OS. It may be necessary to allocate special buffers below 16 megabytes for some 16-bit Host Adapters at this point (see Appendix G).

The driver should make an *AddDiskDevice* call for each new non-registered device which it encounters on the adapter card. Devices that are already registered with the OS should not be queried or examined by this routine. When all attached devices have been accounted for or examined, the driver should return control to the caller. **It is important that all devices on the bus be registered with the OS.** If devices are detected that are busy or have not fully initialized, the driver should block the *ScanForDevices* routine to give them sufficient time to do so. This can be done by calling the *CRescheduleLast*, *CYieldIfNeeded* or *CYieldWithDelay* API. If the device remains busy after a sufficient amount of time, the driver may timeout and return to the caller.

Immediately after calling the *AddDiskDevice* routine to add a new device found during the scan, the driver must perform any driver housekeeping to link up the newly created *DiskStructure,* etc.. I/O requests for the device will begin immediately upon return to the *ScanForDevices* caller.
Drivers may elect to call *ScanForDevices* at the end of the *InitializeDriver* routine and ignore all other calls to the routine. Devices may not be dynamically added if the driver is designed this way.

The *ScanForDevices* routine must determine the actual configuration of devices attached to an adapter card. It may be necessary to read the CMOS configuration RAM for internal drives. Drivers handling SCSI adapters may need to interrogate each target address to determine the number and types of devices attached (warning: some drives will lose all factory configuration information if a format command is aborted, including being sent a test unit ready or other command). Special configurations including devices shared on a SCSI bus may dictate that not all devices may be registered, or even scanned. Devices supporting removable media should be registered with the OS even though no media may be present in the device. The OS can subsequently make the device inactive due to failed requests, and it can be later made active by an Activate IOCTL call, followed by a *Return Device Status* IOCTL call which identifies the specific geometry, etc. of the media now present.

Devices which are registered with the OS (by calling *AddDiskDevice*) as a hard disk will immediately be issued an *Activate* IOCTL call by the OS. If the *Activate* call returns successful status back to the OS, the OS will indicate that the device is active and will begin to issue I/O requests to it.

Devices registered with the OS as removable will immediately be issued a *Mount* IOCTL, followed by a *Return Device Status* IOCTL, followed by an *Activate* IOCTL, and finally, followed by a *Lock* IOCTL.

The above IOCTL sequence will occur for a removable device when an NLM makes an *Activate* IOCTL call to the device.

DeleteDevice

```
void DeleteDevice(
    DiskStruct *DiskHandle);
```

where: DiskHandle is the DiskStructure pointer (handle) returned by *a AddDiskDevice* call

NetWare 4.xx & 3.1x (excluding 3.11)

This routine is optional in v4.xx and v3.1x (excluding 3.11) and will normally not be called. Most failed devices are deactivated but remain in memory. However, if the driver detects that the device needs to be removed due to reconfiguration, hardware removal, etc., the DeleteDevice procedure can be called but only at a blocking process level. (This allows the procedure to make blocking calls such as *RemoveDiskDevice* and *DeleteDiskDevice*.)

DeleteDevice must perform the following steps:

1) Force the OS to deactivate the device by putting back with an appropriate error code all requests the drivers has obtained (by issuing a *GetRequest*). This is done by issuing a *PutRequest* with a "Non Media Error" code in the CompletionCode field of the initial request and a "Device Not Active" code in the CompletionCode field of all subsequent requests. Once the device is deactivated, the corresponding elevator queue is emptied and all requests returned with a "Device Not Active" error code.

2) Call *RemoveDiskDevice*. The driver must remain active to receive I/O requests and IOCTL requests until the OS returns from this call. If the device has not been deactivated at this point, the OS will attempt to flush requests to the driver by calling it's *IOPoll* routine. These requests should be obtained and put back (using *GetRequest* and *PutRequest*) with a "Non Media Error" and "Device Not Active" completion codes. The OS returns from *RemoveDiskDevice* when the elevator is empty.

3) Uncouple the *DeviceStructure* and related information from the *CardStructure*.

4) Call *DeleteDiskDevice*.

5) Return.

NetWare 3.11

In v3.11 *DeleteDevice* is registered with the OS though the *AddDiskSystem* call. When the device driver determines that the device should be removed, it informs the OS using an *AlertDevice* call and the "Delete Device" MessageBit parameter. The OS will then deactivate the device, clear all pending requests owned by the OS and call the *DeleteDevice* routine.

DeleteDevice must perform the following steps:

1) Put back with an appropriate error code all requests the driver has obtained (by issuing a *GetRequest*). This is done by issuing a *PutRequest* with a "Device Not Active" code in the CompletionCode field of the requests.

2) Call *RemoveDiskDevice*. The driver must remain active to receive IOCTL and I/O requests until the OS returns from this call.

3) Uncouple the *DeviceStructure* and related information from the *CardStructure*.

4) Call *DeleteDiskDevice*.

5) Return.