8

Hub Management Interface

Introduction

The Hub Management Interface (HMI) specifies how hub management functions are integrated into the ODI model. Novell has currently defined an HMI specification for Ethernet 10BaseT repeaters (based on IEEE's specification for 10BaseT repeater management), and an HMI specification for Token-Ring concentrator management. The HMI architecture is extensible and allows for additional hub types in the future.

There are two possible ways to implement HMI functions in ODI drivers. The developer may implement the HMI capabilities described in this chapter as an additional feature of a standard Multiple Link Interface Driver (MLID), or a driver could be developed that is limited to providing only those functions required by the HMI. In most cases, hub adapters also incorporate a network interface, so a fully functional MLID with management support is the common solution.

The HMI specification allows drivers to manage hubs resident in the server or external as Figure 8.1 shows.

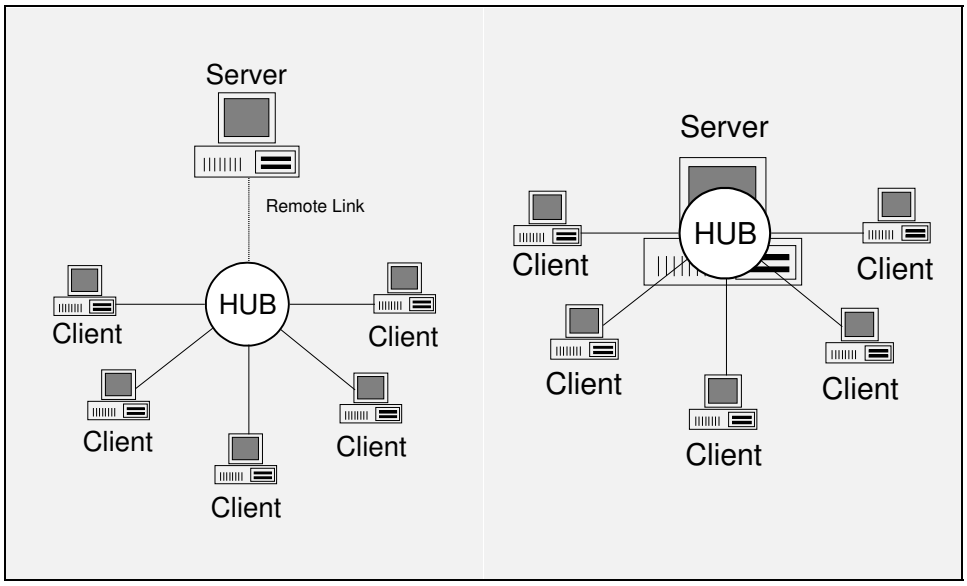

Figure 8.1 External and Internal Hubs

Hub Hardware

An internal hub typically consists of one or more adapters. There may be a master and one or more slave adapters. Each adapter has several ports to which various networked devices are attached. The set of ports on a single adapter is typically called a group. One or more groups operating together constitutes a hub. The minimum configuration for a hub would be a single adapter with a group of ports, normally containing a network controller. A more complex hub would contain several groups (adapters) as shown below in Figure 8.2.

In this chapter, *group* signifies the smallest field replaceable hardware unit -- the adapter. *Hub* refers to a manageable wiring concentrator handling any media type, *repeater* refers to a 10BaseT repeater, and *concentrator* refers to a Token-Ring concentrator.

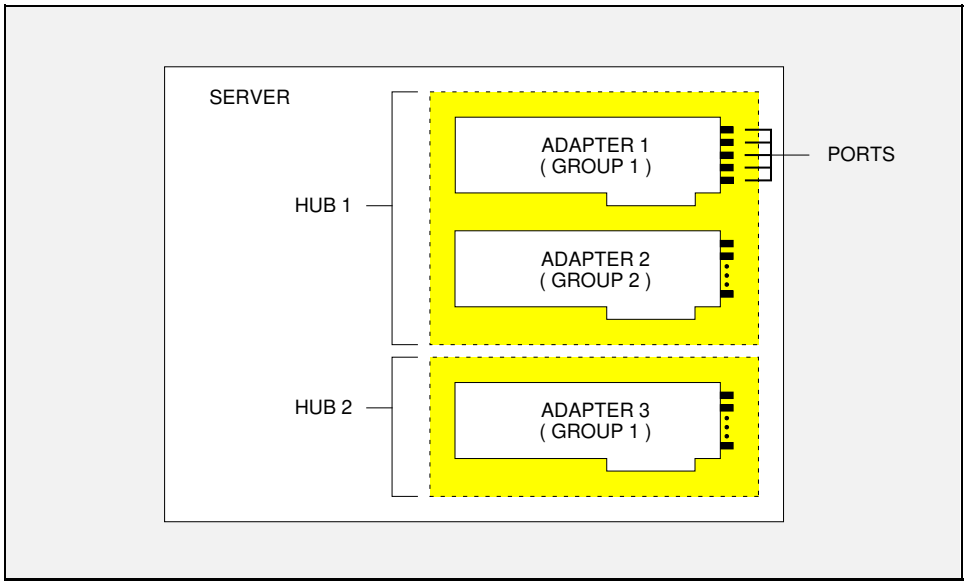

Figure 8.2 Server-Resident Hubs Diagram

In addition to adapters installed in the server, an external hub may also be managed using the HMI. It is possible that an external hub may be linked to the server via a serial port. Special consideration must be given to implementing the HMI functions in the driver if serial port communications are involved so that the driver's operation does not adversely affect the server's performance. Refer to the *Managing External Hubs* section later in this chapter for additional information.

Manageable hub hardware will have some counters and capabilities that the HMI software must access and use. Other HMI-required features may not be provided by the hardware. If the hardware does not explicitly implement some required feature, that feature must be implemented by the software.

Token-Ring Design Considerations

In addition to the HMI functionality described in this chapter, a Token-Ring hub must adhere to the following requirements.

1) The local host port is always the upstream neighbor of the first active port on the hub adapter.

2) The directional flow of the token in the ring must start from the lowest numbered port up to the highest numbered port. This would mean that if port numbers 3, 4, and 5 are in the ring, port number 3 is the upstream neighbor of port 4, while port number 4 is the upstream neighbor of port 5.

   This sequencing holds true between successive adapters as well. The directional flow must start from the lowest numbered adapter up to the highest numbered adapter in the ring.

3) Any external ports in the ring must follow the last active port in the last adapter of the hub. Figure 8.3 shows an example of a network with two hub adapters along with external ports connected via a ring-in or a ring-out port.

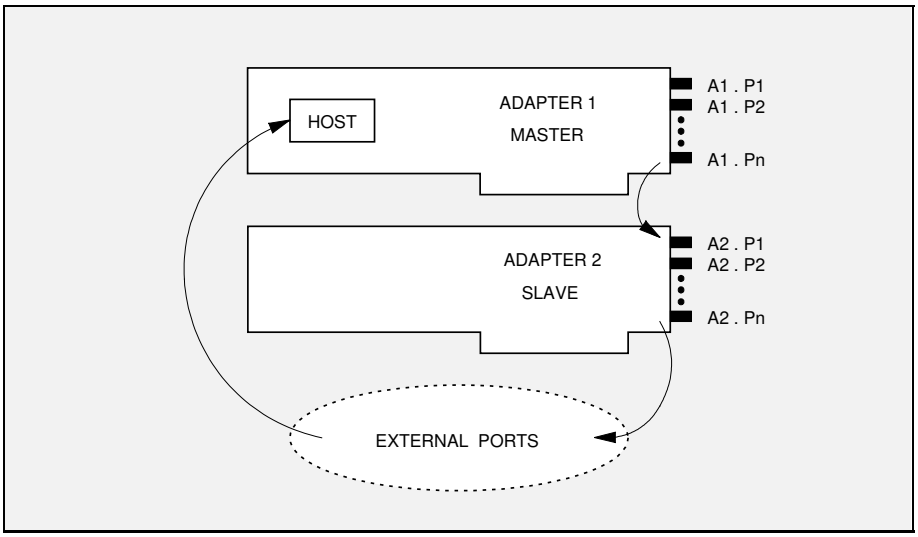

Figure 8.3 Token-Ring configuration of 2 hubs, connected to external ports.

The ring map for this system must conform with the following sequence:

```
HOST, A1(P1...Pn), A2(P1...Pn), EXT(P1...Pn)
```

```
HOST   = local host port
A#     = adapter number
P#     = port number
EXT-P# = external port
```

4) The MAC address of the file server must hold a static position, relative to other ports on the master adapter, reported in the HIT table.

5) Token Ring hubs together can be daisy chained to form a ring. However, they are not expected to form a hierarchical ring. If such implementation is adopted, accuracy of the ring information cannot be guaranteed.

Hub Management Software

Hub management within the ODI model consists of a Hub Management NLM that interfaces with network management agents such as SNMP; the Hub Support Layer (HSL) that provides basic hub management support services; and MLIDs that contain hub management functions that access the hardware. Figure 8.4 shows the hub management interface elements introduced in this chapter in relation to the rest of the ODI model.

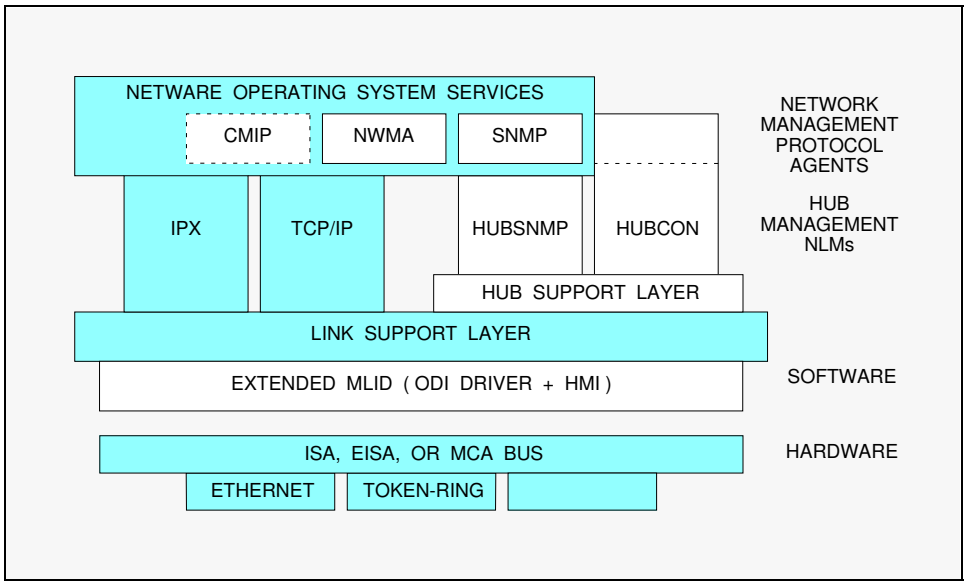

Figure 8.4 Hub Management in the ODI Model

When the network management protocol agent receives a hub management request it routes the request to the hub management NLM. The NLM places the request in a Hub Command ECB and passes it via the HSL / LSL to the appropriate driver. The driver handles the request and returns the ECB.

The majority of communications between the management NLM and the HMI functions in the driver are initiated by the management NLM in the form of hub management requests. The driver generates only a few specific notification messages that it passes up to the HSL. The HSL passes the messages on to the appropriate management NLMs.

Network Management Protocol Agent

The network management protocol agent works with protocol stacks and upper layer applications. It is responsible for processing network management requests from these sources or it may generate management requests itself. The management agent interprets and sends the requests and retrieves requested management data. Examples of management agents are: Simple Network Management Protocol (SNMP), NetWare Management Agent (NWMA), and Common Management Interface Protocol (CMIP).

Hub Management NLM

The hub management NLM (shown in Figure 8.4 as HUBSNMP) is the interface between a server-resident network management protocol agent and the HSL. There may be more than one hub management NLM present in the server at one time. Each one interfaces to a single network management protocol agent and is responsible for providing access to a specific set of management objects to that agent. There may be multiple management agents resident to support various protocols. The NLM formulates requests into Hub Command Event Control Blocks and passes them to the HSL.

Hub Support Layer

The Hub Support Layer (shown in Figure 8.4) acts as a central management node in the hub management system. It performs global maintenance of hub managed information in the form of externalized services available to other hub management NLMs, as well as automatic services implicitly performed to maintain integrity of the hubs. External services include sending messages down to the LSL or routing notification messages up to management NLMs. The automatic services include preservation of port states over power outages.

HUBCON Utility

The Novell-provided utility, HUBCON, is a management NLM that allows the user to control and monitor hubs from the server console. This utility performs local (only for the server it is installed on) management functions, interfaces with the HSL, and displays management information on the console. HUBCON may be used remotely via the NetWare Remote Management Facility (RMF), or the XCONSOLE utility.

Hub Management MLID

The HMI function in the driver is responsible for carrying out hub management requests. Requests require the driver to access hardware or manipulate a software implemented feature. Management requests are either GET or SET functions. GET functions request the values of specific statistics or other registers. SET functions place specific values in control variables or registers, or cause an action, such as a reset or self-test, to occur.

A driver with management support must also indicate any change in the operability or configuration of the hardware by allocating a hub notification ECB, filling it in, and returning it to the hub management NLM. Event types that require notification include: hub health changes, group changes, and hardware resets or failures.

Hub Management Objects

Managed *objects* are state variables, counters, registers, or descriptive data strings used in hub management. IEEE and ISO specifications define managed *objects* in groupings called *packages*.

There are four packages of managed objects defined in this chapter. The first three are packages defined in the IEEE layer management specification: Basic Control, Performance Monitoring, and Address Tracking. In addition, Novell has defined an additional management object package, the Novell Extension package.

The hub management portion of the driver must map the management features of the hardware to the managed objects defined by the HMI specification. Some hardware features will map directly to HMI-defined objects. Other hardware features may not be defined by the HMI specification, or the hardware may not support some capabilities required by the HMI specification.

Developers must support all objects in all packages defined in this chapter in order to claim *full compliance* to the HMI specification. However, only the objects in the Basic Control package are mandatory. You may choose to implement a subset of the management packages (i.e. Basic Control and Performance Management only), but in order to claim compliance to the HMI specification for a particular package, *all objects in that package* must be supported. If the hardware does not explicitly implement some required feature, the driver must implement that managed object in software or return a "not supported" command status for that object. Simply returning a zero value for unsupported objects is not sufficient.

10BaseT Object Packages

There are four packages of managed objects defined by the HMI specification for Ethernet 10BaseT hubs. The first three are packages defined in the IEEE layer management specification: Basic Control, Performance Monitoring, and Address Tracking. In addition, Novell has defined an additional management object package, the Novell Extension package.

Basic Control (defined by IEEE)

- RepeaterID
- RepeaterGroupCapacity
- RepeaterHealth
- RepeaterHealthState
- RepeaterHealthText
- RepeaterHealthData
- RepeaterReset
- RepeaterResetAction
- ExecuteNonDisruptiveSelfTestAction
- GroupID
- GroupPortCapacity
- GroupMap
- GroupMapChange
- PortMap [1]
- PortMapChange [1]
- PortAdminState [2]
- PortAdminControl
- AutoPartitionState

Performance Monitoring (defined by IEEE)

- ReadableFrames - reported per port
- ReadableOctets - reported per port
- FrameCheckSequenceErrors - reported per port
- AlignmentErrors - reported per port
- FramesTooLong - reported per port
- ShortEvents - reported per port
- Runts - reported per port
- Collisions - reported per port and hub
- LateEvents - reported per port
- VeryLongEvents - reported per port and hub [3]
- DataRateMismatches - reported per port
- AutoPartitions - reported per port

Address Tracking (defined by IEEE)

- LastSourceAddress
- SourceAddressChanges

Novell Extensions

- PortLinkState
- RepeaterTotalOctets
- PortType
- StationPortAddress
- PortReceiveControl
- PortTransmitControl
- AutoDisable
- NotifySourceAddrChange

[1] **Note:** These objects, although specified by IEEE, are not required by the HMI specification.

[2] **Note:** The IEEE specification requires that the value of this object be maintained when the power is off. The HSL saves the new port state whenever it changes, then restores the last saved state after the driver and HSL are reloaded. Before the state is restored, during initial booting, the port states are determined by the power-up state of the hardware. The power-up state is determined by the hardware manufacturer.

Installations that have disabled ports for security reasons may find power up with all ports enabled unacceptable. Other installations may reject having the ports power up in the disabled state because of the inconvenience. The optimal situation is to have each port power up in the state that it was last configured. Hub designers should carefully consider these issues when developing their product.

[3] **Note:** The veryLongEvents object replaced the repeaterMJLPs object defined by the IEEE in previous drafts of its specification. Novell has chosen to add this new object definition to the HMI as PortVeryLongEvents in addition to keeping the original object as RepeaterVeryLongEvents.

Token-Ring Object Packages

The HMI specification for Token-Ring hubs requires very limited functionality from the ODI driver. The driver must implement only the Basic Control object package. The additional Performance Monitoring and Address Tracking information necessary for managing Token-Ring hubs is collected and maintained by the Hub Support Layer (HSL). The driver need only enable the Ring Error Monitor and Configuration Report Server so the HSL can receive and collect the Performance Monitoring and Address Tracking information.

Basic Control

- ConcentratorID
- ConcentratorGroupCapacity
- ConcentratorHealth
- ConcentratorHealthState
- ConcentratorHealthText
- ConcentratorHealthData
- ConcentratorReset
- ConcentratorResetAction
- ConcentratorType
- ConcentratorSpeed
- ExecuteNonDisruptiveSelfTestAction
- GroupID
- GroupPortCapacity
- GroupMap
- GroupMapChange
- GroupCableType
- PortMap [1]
- PortMapChange [1]
- PortAdminState [2]
- PortStatus
- PortType

Performance Monitoring

This information is maintained by the HSL for all HMI ports as well as ports external to the hub but on the ring.

- Line Errors - reported per port
- Burst Errors - reported per port
- AC Errors - reported per port
- Abort Transmission Errors - reported per port
- Internal Errors - reported per port
- Duplicate Addresses - reported per port and ring
- Receive Congestions - reported per port and ring
- Beacons Transmitted - reported per port and ring

- Set Recovery - reported per port and ring
- Ring Signal Loss - reported per port and ring
- Monitor Contention Errors - reported per port and ring
- Frequency Errors - reported per ring
- Lost Frame Errors - reported per ring
- Frame Copied Errors - reported per ring
- Token Errors - reported per ring
- Active Monitor Errors - reported per ring
- Duplicate Monitors - reported per ring
- Ring Poll Failures - reported per ring

Address Tracking

This information is collected and maintained by the HSL.

- Active Monitor
- Active Monitor Changes
- Last Upstream Address
- Upstream Address Changes
- Last Source Address
- Source Address Changes

In the Token-Ring HMI, the hub support layer must receive both HMI management event control blocks (ECBs) as well as MAC frames from the driver. Accordingly, the hub support layer first registers with and binds to the driver as a protocol stack using the "HUBMGR" protocol ID. It then registers itself as a default protocol stack in order to receive MAC frames.

It is the responsibility of the driver to enable the Ring Error Monitor and the Configuration Report Server so the HSL can receive the corresponding MAC frames to collect the Performance Monitoring and Address Tracking information. These functions are normally enabled via a protocol stack by calling the driver control procedure *Driver-MulticastChange*, passing the functional address C00000000018h. However, in the case of HMI, the driver is responsible for turning the appropriate functional address bits on.

All request packets sent to the driver are delivered via the raw send facility. When sending a MAC frame to the driver, the hub support layer includes the entire frame starting from the AC bits. Similarly, when receiving, the hub support layer expects the entire MAC frame.

An HMI driver may send any of the legal MAC frames to the HSL. The HSL ignores any MAC frames that it does not require. However, to ensure proper operation, the following MAC frames must be sent to the HSL via the LSL.

- Report Beacon Frame
- Active Monitor Present (optional)
- Standby Monitor Present (optional)
- Report New Monitor
- Report SUA Change
- Report Ring Poll Failure
- Report Monitor Error
- Report Error
- Report Station Address

In addition, the driver must be able to accept and transmit the following MAC frame sent to it by the hub support layer.

- Request Station Address

Object Identifiers

Each set of object identifiers defined for the HMI interface belongs to a specific *domain*. The domain together with specific object identifier values uniquely identify any given object. The domain is a 4-byte character string (without a null-terminating character) that is passed with the command and notification ECBs.

In the following sections, all object identifiers defined by the HMI specification for 10BaseT repeaters belong to the "NVL1" domain. All object identifiers defined by the HMI specification for Token-Ring concentrators belong to the "NVL2" domain.

The tables in this section indicate specific values that the driver must use to define managed objects in order to support the HMI specification.

The tables include:

- Object Name
- Object ID - a unique value associated with the object name and used to access the object. The driver must define the object to have the ID value indicated in this table.
- Object Length (in bytes)
- The possible GET or SET commands for the object. Objects that are counters only support GET commands. The action objects support GET and SET commands as shown in the tables.
- Object Management Class - Indicates whether the object must be managed on a per *hub*, per *group*, or per *port* basis.

10BaseT Object Identifiers

Each set of object identifiers defined in the HMI specification for 10BaseT hubs belongs to the "NVL1" domain. The domain together with the specific object ID values indicated in the following tables, uniquely identifies any given object.

The first table contains those managed objects that are required by the Basic Control package (and are not defined in the information tables described later in this chapter). The remaining tables contain those managed objects defined for Performance Monitoring, Address Tracking, and the Novell Extension packages. All counter objects described in these tables are 32 bits in length.

Basic Control

Object	ID	Length	Get/Set	Class
InfoTablePointer	0	4	Get	Hub
ResetHubAction	1	4	Get/Set	Hub
ExecuteSelfTest1Action	2	4	Get/Set	Hub
ExecuteSelfTest2Action	3	4	Get/Set	Hub
PortAdminState	4	4	Get/Set	Port
AutoPartitionState	5	4	Get	Port

InfoTablePointer. Pointer to the Hub Information Table (HIT). A GET command for this object *always* returns a pointer.

ResetHubAction. A SET request for this object should initiate a reset of the hub system. If the reset is completed before the driver's management routine returns, set the *ObjectValue* to 1. If the reset is still executing on return, set the value to 2. A SET command for this object must always generate a reset notification as described later in this chapter.

The possible values returned by a GET are:

1 - the repeater is not in the process of resetting
2 - the repeater is in the process of resetting

Note: Normally only the reset notification would result from a SET, however, if resetting the repeater results in a health state change, the driver must also generate a HealthChange notification. See the appropriate IEEE document for the reset requirements.

ExecuteSelfTest1Action. A SET request for this object with an *ObjectValue* of 2, causes the repeater to execute a non-disruptive self test. If the self test is completed before the driver's management routine returns, set the *ObjectValue* to 1. If the self test is still executing on return, set the value to 2. If executing the self test results in a health state change, a *HealthChange* notification must be generated as described in the notification section of this chapter. See the appropriate IEEE document for the *ExecuteNonDisruptiveSelfTest* requirements.

The possible GET values are:

1 - non-disruptive self test is not executing
2 - non-disruptive self test is executing

ExecuteSelfTest2Action. A SET request for this object with an *ObjectValue* of 2, causes the repeater to execute a disruptive self test. If the self test is completed before the driver's management routine returns, set the *ObjectValue* to 1. If the self test is still executing on return, set the value to 2. If executing the self test results in a health state change, a *HealthChange* notification must be generated as described in the notification section of this chapter.

The possible GET values are:

1 - disruptive self test is not executing
2 - disruptive self test is executing

PortAdminState. A value indicating the current state of the port. The possible *ObjectValues* for both SET and GET commands are:

1 - Disable; the port is prevented from transmitting or receiving data.
2 - Enable; the port is not disabled. (This value must not be interpreted to indicate the health status of the port.)

Note: The HSL will preserve the PortAdminState when the power is off as required by the IEEE specification. The HSL saves the new port state each time it changes, and restores the last saved state after the driver and HSL are loaded.

AutoPartitionState. The auto partition value indicates whether the port is currently partitioned by the hub's auto-partition algorithm. Typically a port is automatically partitioned if it is signalling a continuous collision or detects some number of consecutive collisions. The possible GET values returned are:

1 - the port is auto-partitioned
2 - the port is not auto-partitioned

Performance Monitoring

Object	ID	Length	Get/Set	Class
TransmitCollisions	6	4	Get	Hub
RepeaterVeryLongEvents	7	4	Get	Hub
PortVeryLongEvents	8	4	Get	Port
ReadableFrames	9	4	Get	Port
ReadableOctets	10	4	Get	Port
FrameCheckSequenceErrors	11	4	Get	Port
AlignmentErrors	12	4	Get	Port
FramesTooLong	13	4	Get	Port
ShortEvents	14	4	Get	Port
Runts	15	4	Get	Port
Collisions	16	4	Get	Port
LateEvents	17	4	Get	Port
DataRateMismatches	18	4	Get	Port
AutoPartitions	19	4	Get	Port

TransmitCollisions. The number of transmit collisions detected by the repeater. This counter is incremented once for each collision detected.

RepeaterVeryLongEvents. The total number of packets received by all ports in the repeater that were too long causing the associated ports to enter the jabber protection condition. Other counters may also be incremented as required.

PortVeryLongEvents. The number of packets received by the port that were too long causing the port to enter the jabber protection condition. Other counters may also be incremented as required.

ReadableFrames. The number of valid length frames received without FCS errors or collisions. Does **not** include invalid frames of any type.

ReadableOctets. The number of octets received in the frames counted in *ReadableFrames*.

FrameCheckSequenceErrors. The number of valid length frames received with FCS errors and without Framing errors or collisions.

AlignmentErrors. The number of alignment errors detected by the port. This counter is incremented for each valid length frame received with both FCS AND Framing errors and without collisions. If this counter is incremented, then do not increment the FCS error counter for the same frame.

FramesTooLong. The number of frames longer than 1518 bytes (including the FCS) received by the port. All long frames should be counted in this total, whether the FCS is correct or not. However, if this counter is incremented, then do not increment the AlignmentErrors or FrameCheckSequenceErrors counters.

ShortEvents. The number of short fragments received by the port. A short fragment is a carrier event of less than 7.4 microseconds. *ShortEvents* may indicate externally generated noise hits which will cause the repeater to transmit Runts to its other ports, or propagate a collision (which may be late) back to the transmitting DTE and damaged frames to the rest of the network.

Runts. The number of runt fragments detected by the port. A runt is a frame with less than 63 bytes and more than 3 bytes, including the FCS, but not including 8 bytes of preamble. All runts should be counted whether the FCS is correct or not.

Collisions. The number of collisions detected by the port. This counter includes the late collisions counted in *LateEvents*.

LateEvents. The number of late collisions detected by the port. A late event is a carrier event which qualifies as a collision at any time when the "ActivityDuration" is greater than 56 microseconds. This event is counted in both *Collisions* and *LateEvents*.

DataRateMismatches. The number of times a frame has been received by the port with the transmission frequency (i.e. data rate) detectably different from the specified data rate. Other counters may also be incremented as required.

AutoPartitions. A counter of the number of times the repeater has auto-partitioned this port. An auto-partition is caused by an excessive number of consecutive collisions (30) or excessive duration of collision (100 microseconds). When either of these events occur, the hub automatically disables the port for reception. The port will be re-enabled automatically after a single good reception from the port. The partition status is also reset to not-partitioned after manual disabling and re-enabling of the port.

Address Tracking

Object	ID	Length	Get/Set	Class
LastSourceAddress	20	6	Get	Port
SourceAddressChanges	21	4	Get	Port

LastSourceAddress. The source address from the last frame received by the port that meets the criteria for the *ReadableFrames* object.

SourceAddressChanges. The number of times the *LastSourceAddress* changes value. Only address changes for frames that meet the criteria for the *ReadableFrames* object should be counted.

Novell Extensions

Object	ID	Length	Get/Set	Class
PortLinkState	22	4	Get	Port
RepeaterTotalOctets	23	4	Get	Hub
PortType	24	4	Get	Port
StationPortAddress	25	6	Get/Set	Port
PortReceiveControl	26	4	Get/Set	Port
PortTransmitControl	27	4	Get/Set	Port
AutoDisable	28	4	Get/Set	Port
NotifySourceAddrChange	29	4	Get/Set	Port

PortLinkState. A value indicating the current state of the link attached to the port. The possible values are:

1 - LinkDown, the link pulses are not being received by the port
2 - LinkUp, the link pulses are being received by the port
3 - NotApplicable, the port is an AUI or other non-10BaseT port

RepeaterTotalOctets. A value indicating the total number of bytes repeated including 8 bytes of preamble and FCS for any packet, whether the FCS was correct or not.

PortType. A value indicating the type of port. The possible GET values returned are:

1 - Other
2 - Normal port
3 - Local host port
4 - AUI port

StationPortAddress. A GET request for this object returns the MAC address set for this object. If specified, this MAC address will have a non-zero value corresponding to the MAC address that should be connected to the port. The value from this object is used in conjunction with the *PortReceiveControl* and *PortTransmitControl* objects to verify security. A SET command will change this object's value to the specified MAC address.

PortReceiveControl. A SET of this object with an *ObjectValue* of 1 allows the port to receive any packet whether or not they are destined to this port. A SET with an *ObjectValue* of 2 causes all packets received by this port that are not destined for the port to be jammed. A packet is destined for a port if:

a) The destination address of the packet matches the *StationPort-Address* of the port.
b) The packet is a broadcast or a multicast.

A GET request returns the current state of this object as follows:

1 - *PortReceiveControl* is not enabled; the port can view all incoming packets.
2 - *PortReceiveControl* is enabled; all packets not destined to this port are jammed.

PortTransmitControl. A SET of this object with an *ObjectValue* of 1 allows the port to transmit packets to any station in the network. A SET with an *ObjectValue* of 2 allows the port to transmit packets only when the source address is the same as the *StationPortAddress*. If a packet violating this condition is encountered and the *AutoDisable* object is set, this port will be disabled. A GET request returns the current state of this object as follows:

1 - *PortTransmitControl* is not enabled; the port can transmit to any station in the network.
2 - *PortTransmitControl* is enabled; only packets with a source address equal to the *StationPortAddress* will be transmitted.

AutoDisable. A SET of this object with an *ObjectValue* of 2 allows the driver to automatically disable a port that has violated the condition specified in *PortTransmitControl*. A SET with an *ObjectValue* of 1 means the driver should not automatically disable a port. A GET request returns the current state of the *AutoDisable* object as follows:

1 - Port will not be auto disabled. (AutoDisable mode is off).
2 - Port can be auto disabled. (AutoDisable mode is on).

NotifySourceAddrChange. A SET of this object with an *ObjectValue* of 2 enables notifications on this port whenever the last source address for the port changes. A SET with an *ObjectValue* of 1 disables notifications. A GET request returns the current state of this object as follows:

1 - Last source address change notifications will not be transmitted for this port.
2 - Last source address change notifications will be transmitted for this port.

Token-Ring Object Identifiers

Each set of object identifiers defined in the HMI specification for Token-Ring concentrators belongs to the "NVL2" domain. The domain combined with the specific object ID values indicated in the following tables, uniquely identifies any given object.

The table below contains those managed objects that are required by the Basic Control package (and are not defined in the information tables described later in this chapter).

Basic Control

Object	ID	Length	Get/Set	Class
InfoTablePointer	0	4	Get	Hub
ResetHubAction	1	4	Get/Set	Hub
ExecuteSelfTest1Action	2	4	Get/Set	Hub
PortAdminState	3	4	Get/Set	Port
PortStatus	4	4	Get	Port
PortType	5	4	Get	Port

InfoTablePointer. Pointer to the Hub Information Table (HIT). A GET command for this object *always* returns a pointer to the HIT.

ResetHubAction. A SET request for this object should initiate a reset of the hub system. If the reset is completed before the driver's management routine returns, set the *ObjectValue* to 1. If the reset is still executing on return, set the value to 2.

The possible *ObjectValues* returned by a GET are:

1 - the concentrator is not in the process of resetting
2 - the concentrator is in the process of resetting

A SET command for this object must always generate a reset notification as described later in this chapter. Normally only the reset notification would result from a SET, however, if resetting the repeater results in a health state change, the driver must also generate a HealthChange notification.

Performing the reset function must not alter the normal operation of the ring in any way.

ExecuteSelfTest1Action. A SET request for this object with an *ObjectValue* of 2, causes the concentrator to execute a non-disruptive self test. If the self test is completed before the driver's management routine returns, set the *ObjectValue* to 1. If the self test is still executing on return, set the value to 2.

The possible GET values are:

1 - non-disruptive self test is not executing
2 - non-disruptive self test is executing

If executing the self test results in a health state change, a HealthChange notification must be generated as described in the notification section of this chapter.

Note: This test is to be defined by the hardware vendor. However, performing the self test function may not alter the normal operation of the ring in any way.

PortAdminState. A value indicating the current state of the port. This object can be used to enable or disable the Trunk Port Ring-In and Ring-Out ports as well as all other regular ports. The possible *ObjectValues* for both SET and GET commands are:

1 - Disable; the port is prevented from transmitting or receiving data.
2 - Enable; the port is not disabled (This value must not be interpreted to indicate the health status of the port.)

Note: The HSL will preserve the PortAdminState when the power is off as required by the IEEE specification. The HSL saves the new port state each time it changes, and restores the last saved state after the driver and HSL are loaded.

PortStatus. A value indicating the current status of the device attached to the port. The possible GET values returned are:

1 - Phantom current present (normal port)
2 - Phantom current not present (normal port)
3 - Internally wrapped (ring-in/ring-out port)
4 - Not internally wrapped (ring-in/ring-out port)

PortType. A value indicating the type of port. The possible GET values returned are:

1 - Other
2 - Ring-in port
3 - Ring-out port
4 - Daisy-in port
5 - Daisy-out port
6 - Normal port

Information Tables

Many of the managed objects described in the previous sections are defined by the driver in the Hub Information Table (HIT) and the Group Information Table (GIT). Figure 8.5 shows the 1-to-1 correspondence of hubs with Hub Information Tables, and adapters with Group Information Tables. The following sections describe these structures in detail.

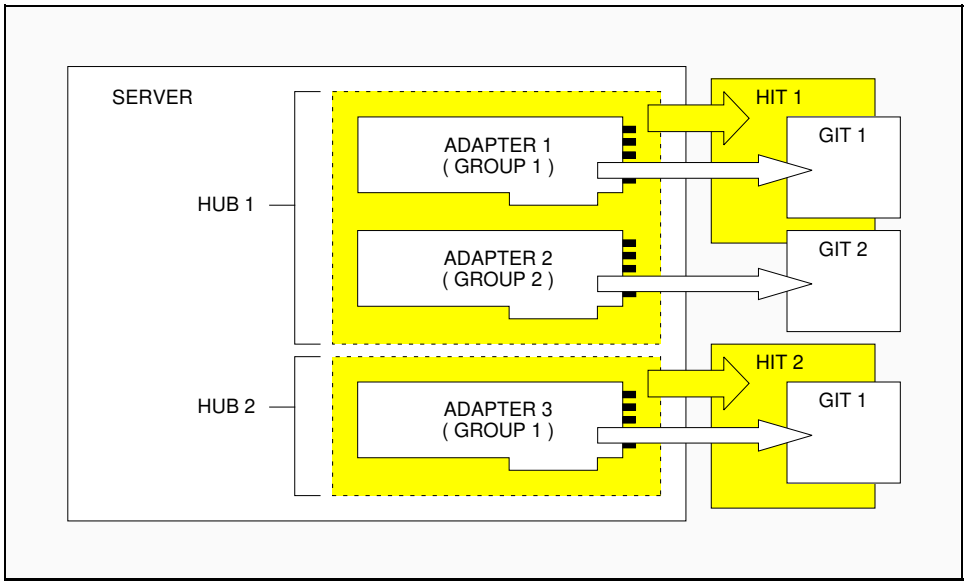

Figure 8.5 Required Instances of the HIT and GIT

Hub Information Table

The Hub Information Table (HIT), shown below, is a data structure allocated once for each hub managed. The HMI driver should allocate the HIT in the adapter data space.

```
 Reserved                    db 16 dup (0)  ;00h
 HubType                     db ?           ;10h
 Reserved1                   db (0)         ;11h
 MajorVersion                db ?           ;12h
 MinorVersion                db ?           ;13h
 ManufacturerID              db 3 dup (?)   ;14h
 Reserved2                   db (0)         ;17h
 HubDescriptionPointer       dd ?           ;18h
 HubVersionPointer           dd ?           ;1Ch
 HealthState                 dd ?           ;20h
 HealthTextPointer           dd ?           ;24h
 HealthDataPointer           dd ?           ;28h
 HealthDataLength            dw ?           ;2Ch
 GroupsSupported             dw ?           ;2Eh
 GroupInfoTablePointer       dd ?           ;30h
 CapabilitiesBitMap          dd ?           ;34h


(the following fields exist only for Token-Ring HMI)

 ConcentratorType            dw ?           ;38h
 ConcentratorSpeed           dw ?           ;3Ah
```

Hub Information Table Field Descriptions

Offset	Name	Bytes	Description
00h	Reserved	16	This field is reserved and should not be modified by the driver. The HMI should initialize these bytes with zeroes.
10h	HubType	1	This field identifies the type of hub. Valid hub-type values are: 1 - 10BaseT repeater 2 - Token Ring concentrator
11h	Reserved1	1	This field is reserved for future use and should be initialized to zero.
12h	MajorVersion	1	This field contains the major version number of the HIT. Currently this value must be 1.
13h	MinorVersion	1	This field contains the minor version number of the HIT. Currently this value must be 0.
14h	ManufacturerID	3	This field contains an identification number that uniquely identifies the hardware manufacturer. Refer to Draft Recommended Practice IEEE 802.1F/D7 May 1, 1991 8.2, ResourceInfo.ManufacturerID for information on this number. Typically, this number is the first 3 bytes of the 802.3 MAC address.
17h	Reserved2	1	This byte should be initialized with zero.
18h	HubDescriptionPointer	4	A pointer to a zero terminated ASCII string that describes the hub.
1Ch	HubVersionPointer	4	A pointer to a zero terminated ASCII string that describes the hub version.
20h	HealthState	4	This field provides the *HealthState* managed object required for the Basic Control package. It contains a value that identifies the current state of the hub. Valid values are: 1 - undefined or unknown state 2 - OK, no known failures 3 - repeater/concentrator related failure detected 4 - group related failure detected 5 - port related failure detected 6 - general failure (unspecified type) detected

Hub Information Table Field Descriptions
(continued)

	HealthState (continued)		In the case of multiple failures this field must contain the value of the highest priority failure. The priority assignments in high-to-low order are: repeater/concentrator failure group failure port failure general failure
24h	HealthTextPointer	4	A pointer to a zero terminated ASCII string that describes the current health state of the repeater or concentrator. In the case of multiple failures the string describes the highest priority failure. The priorities are the same as in the *HealthState* description above. The *HealthText* may be used as a mechanism to provide detailed failure information or instructions for problem resolution. The text strings are manufacturer specific. *Note:* This field satisfies the Basic Control object requirement for *HealthText*.
28h	HealthDataPointer	4	A pointer to a string of data bytes that describes the operational state of the repeater/concentrator. The *HealthData* may be used as a mechanism to provide detailed failure information or instructions for problem resolution. The encoding of the data for this field is manufacturer specific. *Note:* This field satisfies the Basic Control object requirement for *HealthData*.
2Ch	HealthDataLength	2	The length, in bytes, of the *HealthData* string.
2Eh	GroupsSupported	2	The number of groups that are installed in the hub. This field fulfills the requirement for *GroupCapacity*. In environments where groups can be added or removed with power on, this field contains the maximum number of groups in the hub.
30h	GroupInfoTablePointer	4	A pointer to the Group Information Table. This field fulfills the requirement for *GroupPortCapacity* of the Basic Control objects. The Group Information Table is defined following this Hub Information Table description.

Hub Information Table Field Descriptions
(continued)

34h	CapabilitiesBitMap	4	A bit map indicating which object groups are supported by the hub. The bits are defined as follows: 10BaseT repeaters (domain "NVL1"): (LSB) 0 - Basic Control 1 - Performance Monitoring 2 - Address Tracking 3 - Novell Extensions Token-Ring concentrators (domain "NVL2"): (LSB) 0 - Basic Control *Note:* All "NVL2" performance monitoring and address tracking objects are maintained by the HSL.
38h	ConcentratorType	2	This field is present only for the Token-Ring HMI. It identifies the type of Token-Ring concentrator. Possible values are: 1 - Other 2 - Retiming 3 - Non-retiming
3Ah	ConcentratorSpeed	2	This field is present only for the Token-Ring HMI. It identifies the speed of the Token-Ring concentrator. Possible values are: 1 - Other 2 - 4 Mbits/sec 3 - 16 Mbits/sec

Group Information Table

The HMI driver must maintain a Group Information Table (GIT) for each group supported in the hub. The format of the table is shown below. The GIT fields are described on the following page.

All GITs for a particular hub must be in contiguous memory. Therefore, the driver must allocate sufficient memory for the maximum number of groups in the hub.

In the case of a group that may sometimes be dormant (not part of the hub), the driver must allocate a GIT for that group whether that adapter is active or not. Examples of dormant groups would be additional slave adapters that might be added to a master adapter, or slave adapters that are installed but are inactive. If groups cannot be added or removed dynamically, only active adapters need a GIT allocated.

```
 Installed        db  ?             ;0h
 Slot             db  ?             ;1h
 Ports            dw  ?             ;2h
 Description      dd  ?             ;4h
 ObjectIDPointer  dd  ?             ;8h
 ObjectIDLength   dw  ?             ;Ch
 InstalledTime    dd  ?             ;Eh
 DriverWorkspace  db  10 dup  (?)   ;12h

(the following field exists only for Token-Ring HMI)

 CableType        dw  ?             ;1Ch
```

Group Information Table Field Descriptions

Offset	Name	Bytes	Description
00h	Installed	1	The driver places a 1 in this field at initialization if the corresponding group is installed. If the group is removed from the hub while the system is in operation, the driver places a 0 in this field to indicate that the group is no longer active. If the group is re-installed, the field is again set to 1. Whenever the value of this field changes, the driver must generate a *GroupChange notification*. *Note:* This field supplies the *GroupMap* object requirements of the Basic Control package.
01h	Slot	1	The slot number is not mandatory but the driver developer is encouraged to provide meaningful information for this field. If the slot number is not known, place 0FFh in this field.
02h	Ports	2	The number of ports supported by the group. *Note:* This field supplies the *GroupPortCapacity* object requirement of the Basic Control package.
04h	Description	4	A pointer to a zero terminated ASCII string that completely identifies the group, including full name, version, hardware type, and any other required information.
08h	ObjectIDPointer	4	A pointer to an ObjectIdentifier in the form of a 16-bit word array that is the manufacturer's authoritative identification of the group. This value is allocated in the Structure and Identification of Management Information for TCP/IP-based Internets (refer to the document of that name listed at the end of this chapter) enterprises subtree and provides a straight-forward and unambiguous means for determining what kind of group is being managed.
0Ch	ObjectIDLength	2	The length, in bytes, of the ObjectIdentifier.
0Eh	InstalledTime	4	The value returned by the *GetCurrentTime* system call when the group was last installed in the hub.
12h	DriverWorkspace	10	Reserved for driver use.
1Ch	CableType	2	This field exists only for the Token-Ring concentrator HMI. It contains a value indicating the cable type of the associated group. The possible values are: 1 - Other 2 - STP 3 - UTP 4 - Fiber

Note: If the *Installed* field is non-zero, indicating that the group is installed and active, then all of the fields GIT must contain meaningful information.

Hub Management MLIDs

The HMI function in the driver is responsible for carrying out hub management requests. Requests require the driver to access hardware or manipulate a software implemented feature. In either case, the driver responds with a completion status. The reset request also requires that the driver generate a notification.

Management requests are either GET or SET functions. GET functions request the values of specific statistics or other registers. SET functions place specific values in control variables or registers, or cause an action, such as a reset or self-test, to occur.

A driver with management support must also indicate any change in the operability or configuration of the hardware by allocating a hub notification ECB, filling it in, and returning it to the hub management NLM. A notification event occurs when a group is added or removed from the hub or when the hardware is reset or has failed.

Because there may be multiple management NLMs loaded on the server, the driver's hub management capability must be able to process commands from several different sources at the same time.

The developer has some choice in what level of hub management capability to provide beyond the required basic control. Novell recommends that all specified features be supported. When it is not possible to be fully compliant with the specification, the driver must indicate lack of support for whatever features are affected as explained.

There are two possible ways to implement HMI functions in ODI drivers. The developer may implement the HMI procedures as an additional feature of a standard MLID, or a driver could be developed that is limited to providing only those functions required by the HMI.

Implementing HMI Support in MLIDs

If the LAN and hub hardware must use the same interrupt, port, or memory resources, a fully functional driver with HMI support should be developed.

Drivers supporting hub management must set bit 8 of the *MLIDFlags* field in the configuration table. (*MLIDFlags* is at offset 58h of the table). Setting this bit allows the hub support layer to locate hub drivers and automatically bind to them at run time.

A driver with HMI support must contain all of the routines required by the ODI specification. Three of these routines will contain the additional capability that the HMI requires. The affected routines are *DriverInit*, *DriverISR* or *DriverPoll*, and an additional control routine, *DriverManagement*.

DriverInit. During initialization, the driver must set up and initialize additional data space for the tables described later in this chapter.

DriverISR / DriverPoll. The board service routine, either *DriverISR* or *DriverPoll,* updates any counters not available directly from the hardware and generates the required notifications to the management agent in the event of resets and status changes.

DriverManagement. The MSM checks for a valid management ProtocolID in the hub command ECB then passes the ECB to the hub's *DriverManagement* routine if one is available (indicated by the *DriverManagementPtr* field in the *DriverParameterBlock*). This routine should handle all management requests rather than extracting them from the *DriverSend* routine.

Implementing HMI Only

Hub-management-only drivers may be created by adding the HMI capabilities to a fully functional driver. Any driver routines (or portions of routines) that are not needed can be stubbed out. This HMI-only driver should be loaded specifying the appropriate hardware interface parameters for the hub (IO port, interrupt, memory address, etc.).

Managing External Hubs

Novell recommends that drivers managing external hubs cache the state of the hub in server memory. This will allow management requests to be satisfied without affecting the server's performance. The cache could be updated periodically by polling the remote device to provide data for GET requests. SET requests should be checked for validity prior to being transferred to the remote device and report that the action has occurred without waiting for confirmation from the device (a process similar to lying sends).

The use of the *EventServiceRoutine* to indicate the completion of the request allows the driver to operate in an asynchronous manner if the management request cannot be fulfilled immediately. This capability allows for proxy management of hubs external to the server (for example, a repeater attached to the server via a serial port). In this case, the driver should be prepared to have multiple management requests outstanding at the same time.

HMI Mode Selection

Note: The custom command line options described in this section do not apply to Token-Ring HMI drivers.

Many hardware implementations require the driver to at least partially process every packet on the network to maintain hub statistics. When there are very high packet rates, server CPU utilization to maintain the statistics is significant. When this limitation exists, the driver may use a custom-defined command line option as described below to allow the supervisor to run the hub in either the full-featured mode or in a stripped-down mode.

The driver may define the HUBMGT custom command line keyword with two possible mode settings: STANDARD and BASIC, as described in Chapter 3. The STANDARD mode is the default mode. In this mode, the driver maintains all objects it is capable of supporting. In BASIC mode the driver maintains only the Basic Control objects and any other objects that can be maintained with minimal impact on server CPU loading. Objects that are not supported in the BASIC mode return "not supported" in *ObjectStatus*.

Command Processing

Commands and responses are passed between the hub management NLM and the HMI driver as command blocks. As a matter of convenience and simplicity, these command blocks are packaged in specialized Hub Command ECBs and passed using a mechanism similar to the one used to pass frames between a protocol stack and a driver. Consequently, hub management NLMs look and behave like protocol stacks from the point of view of the driver.

The Hub Command ECB is structured such that each command can access multiple objects from the same hub, group, and port. The ECB is used to pass object values between the management NLM and the driver.

The driver must return an *ObjectStatus* for each object accessed by the Hub Command ECB. The status indicates the results of the operation for each object. For example, if an attempt to access an unsupported object is made, the driver must return the not supported (2) status for that object. A failure when accessing one object does not affect accesses to other objects made in the same Hub Command ECB.

Hub Command ECB

The format of the Hub Command ECB is shown below. The functions of some of the fields are different from the standard ECB format to accommodate required hub command parameters.

```
Link                dd ?            ;00h
BLink               dd ?            ;04h
Status              dw ?            ;08h
ESRAddress          dd ?            ;0Ah
LogicalID           dw ?            ;0Eh
ProtocolID          db 6 dup(?)     ;10h
BoardNumber         dd ?            ;16h
Group               dw ?            ;1Ah
Port                dw ?            ;1Ch
Function            dw ?            ;1Eh
DriverWorkSpace     dd ?            ;20h
ManagerWorkSpace    db 8 dup(?)     ;24h
PacketLength        dd ?            ;2Ch
Domain              db 4 dup (?)    ;30h
Reserved            dw ?            ;34h

ObjectCount         dw ?            ;36h

ObjectIdentifier    dw ?            ;38h
ObjectReserved      db ?            ;3Ah
ObjectStatus        db ?            ;3Bh
ObjectValue         db 8 dup(?)     ;3Ch
  .
  .
  .
```

Hub Command ECB Field Descriptions

Offset	Name	Bytes	Description
00h	Link	4	This field contains a forward link to another ECB. The LSL, hub management NLM, and driver may use this field while the ECB is in their possession.
04h	BLink	4	This field contains a backward link to another ECB and may be used to create a doubly linked list of ECBs.
08h	Status	2	This field should always be set to 0 upon completion of command processing.
0Ah	ESRAddress	4	This field contains a pointer to a service routine which the driver calls when the specified function has been completed.
0Eh	LogicalID	2	Raw send, 0xFFFFh.
10h	ProtocolID	6	Contains the value 'HUBMGR' which is used to identify the ECB as a Hub Command ECB.
16h	BoardNumber	4	The logical board number of the hub to be accessed. There is a 1-to-1 correspondence between drivers with HMI and hubs.
1Ah	Group	2	Group to be accessed. Group numbers are assigned from 0 through the value of GroupsSupported - 1 (the GroupsSupported value is in the HIT).
1Ch	Port	2	Port to be accessed. Port numbers are assigned from 0 through Ports - 1 (Ports is in the GIT).
1Eh	Function	2	This field identifies the command function to be executed: GET (0) or SET (1).
20h	DriverWorkSpace	4	This field is reserved for use by the driver.
24h	ManagerWorkSpace	8	This field is reserved for use by the hub management NLM and must not be modified by either the LSL or the driver.
2Ch	PacketLength	4	Always 0.
30h	Domain	4	This field contains a 4-byte character string that specifies the domain. The domain defines the meaning of the object ID numbers. The following domains are currently defined. "NVL1" domain = 10BaseT repeater objects "NVL2" domain = Token-Ring concentrator objects Other domains may be defined that reuse object ID numbers. This facilitates vendor unique extensions to the HMI.
34h	Reserved	2	This field is always zero.

Hub Command ECB Field Descriptions
(continued)

36h	ObjectCount	2	This field contains the count of the number of objects that must be accessed for this ECB. For each object accessed, there is a set of *ObjectIdentifier, ObjectReserved, ObjectStatus,* and *ObjectValue* fields. This field will never be zero.
38h	ObjectIdentifier	2	This is the object ID number assigned to the managed object to be accessed (refer to the Object Identifier tables earlier in this chapter).
3Ah	ObjectReserved	1	This field is always zero.
3Bh	ObjectStatus	1	The HMI driver returns the result of the requested operation for the specified object in this field. All objects in a single ECB should be processed independently so that a failure for one object does not affect the operations for the other objects. Result status values are: 0 - successful 1 - value was too big for object 2 - specified object is not supported 3 - value was out of range for object 4 - a SET function was specified for a read-only object 5 - all other errors
3Ch	ObjectValue	8	For a SET command, this field contains the object value (placed here by the NLM) that will perform the desired action. In a GET command, the driver fills this field with the value from the object specified in the *ObjectIdentifier* field of the Hub Command ECB. Most object values are only 4 bytes. This field is filled beginning at 3Ch, therefore 4 byte object values do not use bytes 40h thru 43h.
44h	ObjectIdentifier2 ObjectReserved2 ObjectStatus2 ObjectValue2 . . .	2 1 1 8 . . .	If the Hub Command ECB accesses multiple objects (i.e. *ObjectCount* > 1), the *ObjectIdentifier, ObjectReserved, ObjectStatus,* and *ObjectValue* fields are repeated for each object.

Hub Command Sequence

The sequence of events that sends a management command to the driver's HMI function is:

1. A hub management request is received by the network management agent and passed to the appropriate hub management NLM.

2. The hub management NLM examines the request and determines which driver should process the request. The management NLM constructs a Hub Command ECB with *ProtocolID* = HUBMGR, *LogicalID* = 0xFFFFh (raw send), *Domain* = NVL1 or NVL2, and *Function* = SET or GET. The *ObjectValue* (for SET commands) and *ObjectIdentifier* fields are filled in. Because more than one request may be sent in an ECB, there may be multiple *ObjectValue*, *ObjectIdentifier*, *ObjectReserved*, and *ObjectStatus* fields.

3. The Hub Command ECB is passed to the driver via the HSL, LSL, and MSM code paths.

4. The MSM determines whether the ECB is a valid management ECB by examining the *ProtocolID* field. ECBs containing management requests are passed to the *DriverManagement* control routine.

5. A typical *DriverManagement* routine validates the command, processes the request, and sets the *ObjectStatus* field to the appropriate value. If either the *Domain* or *ObjectID* is not supported, the "not supported" command status must be returned.

The pseudocode at the end of this chapter shows a general example of the HMI command processing functions within the driver.

Design Tips and Notes

In most cases management requests will be fulfilled immediately by the driver. Interrupts will be disabled and should remain disabled during request processing. The driver developer must guard against reentrancy if interrupts are enabled.

If a request cannot be completed immediately (within a few hundred microseconds), the driver can schedule an AES callback process to handle the request. This may be true for the Reset and Self-Test actions.

Notification Processing

The driver's HMI function must pass notification up to the hub support layer if there is any change in the operational status of the hub being managed. The hub support layer directs the notification to the management agents registered with the driver.

Notification Types

Notification must be provided for each of the defined changes in status described below. Prior to generating a notification the driver must update the related information in the HIT or the GIT. The following tables list which fields must be updated prior to sending a notification. A description of each notification type follows the tables.

Notification	ID	Table	Domain	Update Fields
HealthChange	1	HIT	NVL1 or NVL2	HealthState HealthTextPointer HealthDataPointer HealthDataLength
GroupChange	2	GIT	NVL1 or NVL2	Installed
HubReset	3	HIT	NVL1 or NVL2	HealthState HealthTextPointer HealthDataPointer HealthDataLength
LoopbackRecovery	4	N/A	NVL2 only	N/A
Reserved	5-9			
SourceAddressChange	10	N/A	NVL1 only	N/A

Health Change Notification

This notification conveys information related to the operational state of the hub. The HMI driver must generate this notification whenever a change in the health state of any part of the hub is detected.

Group Change Notification

This notification is generated whenever a group is removed from or added to the hub while the system is operating. If removal and addition of adapters is not allowed while the system remains in operation, this notification should never be generated.

Reset Notification

The HMI driver must generate this notification at the completion of a hub reset. The reset notification is sent when the hub is reset as a result of a power-on condition or upon completion of a reset request.

LoopbackRecovery Notification

A "LoopbackRecovery" condition is when the normal connection between 2 adapters fails and an alternate route is used for connecting the 2 boards. For example, in a configuration where the ring-in port is connected to the ring-out port (as a redundant measure), if a daisy-in and daisy-out connection break, the ring-in and ring-out ports will be used for transmitting information in the ring. This is a condition performed by the hardware and not in control of the user. However, they must provide for the redundant measure in order for this mechanism to work. If a vendor does not provide support for this feature, there is no need to support the notification message as it will never occur.

SourceAddressChange

This notification is generated whenever a new source address is observed for a given port and the *NotifySourceAddrChange* Novell Extensions object is enabled (set to 2) for that port. The condition that triggers this trap is identical to that of *LastSourceAddress* and *SourceAddressChanges* in the NVL1 domain.

Notification Generation Sequence

A notification is generated in a manner similar to the way a received frame is processed. A receive ECB is obtained, the *BoardNumber* and *ProtocolID* are copied into the ECB along with the *NotificationID,* the Hub Information Table pointer, the *GroupNumber* and *PortNumber*. The ECB is then returned for processing. The ECB is routed to all hub management NLMs bound to the driver. Some fields of the ECB that are normally used by the driver for network frame processing are ignored in notification generation. The space normally defined as the *ImmediateAddress* and *DriverWorkSpace* is used to pass the *NotificationID, GroupNumber, PortNumber,* and *InfoTablePointer*.

Note: The driver must obtain the receive ECB for the notification by calling the *MSMAllocateRCB* routine. To return the notification ECB, the driver should use the *MSMReturnNotificationECB* / *MSMServiceEvents* combination or the *MSMFastReturnNotificationECB* routine. Chapter 7 describes these procedures.

If no ECB is available for the notification, the notification is not made. It is the responsibility of the upper layers to ensure that they receive notification information (using a GET command) if they require it.

Example

```
HubResetNotification  proc
         :
         :
         mov      esi, 4
         call     MSMAllocateRCB          ; Get notification ECB
         :
         :        (Fill in all required notification information)
         :
         mov      esi, ECBPtr             ; Point to the ECB
         MSMFastReturnNotificationECB     ; Return the ECB directly to
         :                                  the management application
         :
```

Notification ECB

The hub notification ECB format is shown below. Those fields that the driver must handle in generating a notification are explained in the paragraphs that follow.

```
  Link                dd  ?           ;00h
  BLink               dd  ?           ;04h
  Status              dw  ?           ;08h
  ESRAddress          dd  ?           ;0Ah
  LogicalID           dw  ?           ;0Eh
* ProtocolID          db  6 dup (?)   ;10h 'HUBMGR'
* BoardNumber         dd  ?           ;16h from config table
* NotificationID      dw  ?           ;1Ah notification ID
* GroupNumber         dw  ?           ;1Ch group affected
* PortNumber          dw  ?           ;1Eh port affected
* InfoTablePointer    dd  ?           ;20h HIT pointer
  ManagerWorkspace    db  8 dup (?)   ;24h
  Reserved            dd  4 dup (0)   ;2Ch
  Domain              db  4 dup (?)   ;3Ch


* The driver must fill in these fields before passing the
  ECB to the LSL
```

Notification ECB Field Descriptions

Offset	Name	Bytes	Description
00h	Link	4	This field contains a forward link to another ECB.
04h	BLink	4	This field contains a backward link to another ECB.
08h	Status	2	This field should not be modified. The LSL uses the Status field to indicate the current state of the ECB.
0Ah	ESRAddress	4	This field should not be modified. The LSL routes the ECB to the proper management protocol by calling the address in this field.
0Eh	LogicalID	2	This field should not be modified. The LSL places the target management protocol's logical number in this field.
10h	ProtocolID	6	The protocol ID that all drivers must use for sending notification ECBs is the six byte string 'HUBMGR'. It should be declared as: `HubProtocolID db 'HUBMGR'`
16h	BoardNumber	4	This is the **logical** board number from the configuration table of the driver generating the notification.
1Ah	NotificationID	2	The driver loads the notification identifier value in this field. The notification ID values are specified in the Notification Types section of this chapter.
1Ch	GroupNumber	2	The group number associated with the notification, if applicable.
1Eh	PortNumber	2	The port number associated with the notification, if applicable.
20h	Reserved	4	This field is reserved. Its value should be initialized to 1. (20h = 01h, 21h..23h = 00h)
24h	ManagerWorkspace	8	This field is reserved for use by the management NLM.
2Ch	Reserved	16	This field is reserved. These bytes should be initialized to zero.
3Ch	Domain	4	This field contains a 4-byte character string that specifies the domain. The domain defines the meaning of the notification ID numbers. The following domains are currently defined: "NVL1" domain = 10BaseT repeaters "NVL2" domain = Token-Ring concentrators Other domains may be defined that use the same notification ID numbers differently. This will allow for vendor-unique extensions to the HMI.

Hub Management Pseudocode

Three driver routines contain the additional capability that the HMI specification requires. The routines are:

- *DriverInit*
- *DriverISR (or DriverPoll)*
- *DriverManagement*

DriverInit

During initialization, the driver must allocate and initialize the additional data space for the tables specified in this chapter.

```
DriverInit  proc
  .
  .
  .
  Set up Hub Information Table
  Allocate memory for Group Information Tables
  Allocate memory for port/group software counters if counters not available in hardware
  Initialize port counters (if implemented in software)
  Initialize Group Information Tables
  Initialize the repeater/concentrator hardware
  .
  .
  .
```

DriverISR or DriverPoll

The board service routine, either *DriverISR* or *DriverPoll,* updates any counters not available directly from the hardware and generates the required notifications to the management agent in the event of resets and status changes.

```
DriverISR  proc
  .
  .
  .
  Update HIT, GIT, and counters

  IF health change
       generate notification
  ENDIF
  .
  .
  .
```

DriverManagement

On Entry

ESI	Pointer to the management ECB containing the request
EBP	Pointer to the Adapter Data Space
EBX	Pointer to the Frame Data Space

On Return

EAX	00000000h = Success; ECB relinquished 00000001h = Success; ECB queued FFFFFF88h = No such handle; ProtocolID not supported

Description

To avoid Management ECBs from getting queued in the send queue, this control routine has been added to handle all management requests (rather than extracting them from the *DriverSend* routine).

The MSM checks the *ProtocolID* field in the ECB. If the first byte is an ASCII letter greater than 40h, it is a valid management *ProtocolID*. The MSM will then pass the ECB to the hub's *DriverManagement* routine if one is available (indicated by the *DriverManagementPtr* field in the *DriverParameterBlock*).

The *DriverManagement* routine should scan the whole *ProtocolID* to verify that the management request is valid before processing it. (the *ProtocolID* is "HUBMGR" for Hub management requests.) The routine must also verify that each requested *Domain* / *ObjectID* combination is supported. Once the request is processed, the *ObjectStatus* field should be set to the appropriate value (see the "Hub Command ECB field descriptions" earlier in this chapter).

If the routine must respond asynchronously to the management request, the driver should queue the ECB (in this case, the driver must manage its own queue) and return a status of 00000001h in EAX. When the queued request is complete, the driver must make a call to the event service routine specified in the *ESRAddress* field of the ECB as follows:

```
mov   esi, PtrToECB     ;get ptr to command ECB
push  esi               ;pass on stack
call  [esi].ESRAddress  ;call Event Service Routine
add   esp, 4            ;clean up stack
```

DriverManagement Pseudocode

```
IF ProtocolID is not valid

    RETURN FFFFFF88h


FOR each object


    IF Domain-ObjectID combination is not supported

        ObjectStatus = 2  (not supported)

    ELSE

        IF SET function

            IF legal value
                ObjectStatus = 0  (success)
                set specified object
            ELSE
                ObjectStatus = appropriate error code
            ENDIF

        ELSE

            ObjectStatus = 0  (success)
            ObjectValue = get specified object value

        ENDIF

    ENDIF

CONTINUE to next object



RETURN 0
```

References

ISO 7498:1984, Information Processing Systems - Open Systems Interconnection - Basic Reference Model

ISO 7498-3:1989, Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 3:Naming and Addressing

ISO 7498-4:1989, Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 4:Management Framework

ISO 8824:1990, Information Technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)

ISO 8824:1990, Information Technology - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)

ISO/IEC 8802-3:1990, Information Processing Systems - Local Area Networks - Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications

ISO/IEC 10040, Information Processing Systems - Open Systems Interconnection - Systems Management Overview (to be published)

ISO/IEC 10164-1, Information Processing Systems - Open Systems Interconnection - Management Information Services - System Management - Part 1:Object Management Function

ISO/IEC 10165-1, Information Processing Systems - Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 1:Management Information Model

ISO/IEC 10165-2, Information Processing Systems - Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 2:Definition of Management Information

ISO/IEC 10165-4, Information Processing Systems - Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 4:Guidelines for the Definition of Managed Objects (to be published)

IEEE Standard 802-1990 Overview and Architecture

IEEE Standard 802.3i - 1990, System Considerations for Multisegment 10Mb/s Baseband Network (section 13) - Twisted-Pair Medium Attachment Unit (MAU) and Baseband Medium, Type 10BaseT (section 14)

Draft Supplement to IEEE Standard 802.3, Layer Management for 10Mb/s Baseband Repeaters, Section 19

Draft Recommended Practice IEEE 802.1F, Guidelines for the Development of Layer Management Standards

K. McCloghrie and D. McMaster, Definitions of Managed Objects for IEEE 802.3 Repeater Devices, Internet Draft Request for Comments, July, 1991

M. Rose and K. McCloghrie, Structure and Identification of Management Information for TCP/IP-based Internets, Internet Working Group Request for Comments 1155, May 1990