Chapter 6 Technical Reference for NWPA Data Structures

This chapter is a technical reference for data structures used by CDMs and HAMs. The following is a list of the structures described in this chapter:

AttributeInfoStruct

Used by: CDM

Description: The **AttributeInfoStruct** is a structure that the CDM uses to store device-attribute information for a device (or devices) the CDM manages. A copy of this information is passed to the Media Manager when the CDM registers an attribute by calling **CDI_Register_Object_Attribute()**.

A CDM can register multiple attributes, one attribute for each call to **CDI_Register_Object_Attribute()**. The CDM is expected to maintain an instance of this structure for each attribute it registers.

By registering device attributes with the Media Manager, the CDM can present specific information about a device's operational modes to the application layer. For example, a tape CDM can inform an application that its tape device supports multiple blocksizes.

For more information about attributes, refer to the technical reference information on the **CDI_Register_Object_Attribute()** API found in Chapter 7.

Syntax:

```
struct AttributeInfoStruct
    {
        LONG attributeID;
        LONG attributeType;
        LONG attributeLength;
        BYTE attributeName[64];
    };
```

Parameters: *attributeID* This is a 4 byte field containing a unique ID for the attribute being registered. Currently, the NWPA defines the following attribute IDs:

```
0x44454D0A Media Type
0x5241430E Cartridge Type
0x494E5509 Unitsize
0x4F4C420A Blocksize
0x50414308 Capacity
0x4552500E Preferred Unitsize
0x4D455209 Removable Device
0x41455209 Read Only Device
0x50415412 Tape Position Size
0x5041540F Tape Media Size
0x50415411 Tape Write Format
0x50415410 Tape Read Format
0x4E494D12 Minimum Blocksize
0x58414D12 Maximum Blocksize
0x54414415 Data Compression Information
```

attributeType This is a 4 byte field indicating the data-type of the *InfoBuffer* parameter for the get/set entry points associated with the attribute being registered through **CDI_Register_Object_Attribute()**. The data types are defined as follows:

```
0x00000001    String
0x00000002    BYTE
0x00000003    WORD
0x00000004    LONG
0x00000005    Other:    Indicates that the calling
                        application knows what data type to
                        expect from the target CDM.
```

attributeLength This is a 4 byte field containing a value that indicates the byte-length of the *infoBuffer* input parameter to the get/set entry points associated with the attribute. These entry points are registered during **CDI_Register_Object_Attribute()** along with the attribute.

attributeName This is a 64 byte field containing a byte-length-preceded string. The string contains the ASCII codes that make up the name of the attribute being registered, and it is also NULL terminated.

CDMMessageStruct

Used by: CDM

Description: The **CDMMessageStruct** is a data packet containing a control or I/O request from the Media Manager (CDM Message). The **CDMMessageStruct** is identical to the Media Manager internal message structure. The fields in **CDMMessageStruct** contain the pertinent information required to build a control or I/O request. A pointer to this structure is then passed to the CDM which processes the **CDMMessageStruct** and converts it into a HACB request that is compatible with the adapter supporting the desired device.

Syntax:

```
struct CDMMessageStruct
{   LONG msgPutHandle;
    LONG function;
    LONG parameter0;
    LONG parameter1;
    LONG parameter2;
    LONG bufferLength;
    void* buffer;
    LONG cdmReserved[2];
} ;
```

Parameters: *msgPutHandle* This is a 4 byte field containing a handle to the current I/O request issued by the Media Manager. The Media Manger generates this value and uses it to track a request through different execution stages. This field value is needed as an argument for many of the APIs described in this manual, and it should never be altered.

function This is a 1 LONG field. The upper WORD contains control attributes set by the Media Manager for the I/O request, and the lower WORD contains a Media Manager function code set by an application.

For processor independence reasons, the CDM should use the following macros to extract information from this field:

```
#define GET_MSW (function) ((function >> 16) & 0xFFFF)
#define GET_LSW (function) (function & 0xFFFF)
```

Media Manager control and I/O requests are equated to unique hexadecimal function codes (0x0000 - 0x0047). A Media Manager application makes an I/O request by calling a Media Manager API. The application selects a desired I/O action by passing one of the Media Manager function codes as an input parameter. In turn, the Media Manager packages the request into a CDM Message (*CDMMessageStruct*) placing the function code in the lower WORD of this field, and then issues the CDM Message to the target CDM.
The CDM maps this code into a call to one of its locally-implemented control or I/O routines designed to build the corresponding SHACB request. A list of CDM Message types, their corresponding Media Manager function codes, and their corresponding request descriptions can be found in

Chapter 9. As previously mentioned, in building the CDM Message the Media Manager places control attributes associated with the request in the upper WORD of this field. Most of these attributes only have meaning to the Media Manager and OS. The attributes that do have meaning to a CDM are defined as follows:

```
#define SCATTER_ON_BIT  0x0080     Indicates that the request
                                   is in the NWPA's
                                   scatter/gather format. To
                                   inform the HAM, the CDM
                                   must set the
                                   Scatter/Gather_Flag in the
                                   corresponding HACB.

#define HARDWARE_VERIFY_BIT 0x0100    Tells CDMs that they
                                      must set the verify bit
                                      for all write commands.

#define CACHE_OKAY      0x8000     Indicates that controller
                                   and/or device level caching
                                   is okay. If this bit is not
                                   set, all write commands
                                   must write-through any
                                   controller/device caches.
```

bufferLength This is a 1-LONG field. Typically, its value indicates the size of the *buffer* field. However, its content depends on whether or not the request is in scatter/gather format. If it is in scatter/gather format, this field contains the number of entries in the scatter/gather request list. If it is not in scatter/gather format, this field contains the length, in bytes, of the data buffer. This field is set to zero for requests that do not require the movement of data.

buffer This is a 4 byte field of type pointer to void. Typically, the pointer points to the CDM Message's data buffer. However, the structure of the buffer it points at depends on whether or not the request is in scatter/gather format. If it is in scatter/gather format, this field contains the virtual starting address of the scatter/gather request list. The scatter/gather list is generated by the NWPA or a Media Manager application. If the request is not in scatter/gather request, this field contains the virtual address to the data buffer, in which case if the operation is a read, this buffer is where the data is read to. If the operation is a write, this buffer is where the data is read from. This field is set to zero for requests that do not require the movement of data.

> **Note:** For information about the format of a scatter/gather list, refer to section 3.4 of Chapter 3.

cdmReserved This is a 2-LONG field for the private use of the CDM that queues the current CDM Message using **CDI_Queue_Message()**. The

intended use of this field is to allow the CDM to create links between the queued, current message and other CDM messages or HACBs. If the CDM did not explicitly queue the message, it cannot expect the value it placed in this field to persist.

DeviceInfoStruct

Used by: CDM and HAM

Description: This structure contains specific information about a device attached to a host adapter bus. The HAM maintains an instance of this structure for each device it supports and is responsible for filling in field information when it receives a "Scan for New Devices" command issued from the command line. The HAM determines information for some of the fields by probing the hardware (such as unitNumber, busID, etc.). The information for the remaining fields (such as deviceHandle) is generated by the HAM. The HAM uses the information in this structure to report a device and set its attributes. The CDM uses this structure to obtain device information to determine if it will bind to the device. When a device comes online that is of the type for which a CDM has registered, the Media Manager calls that CDM's *CDM_Inquiry()* passing it a pointer to this structure. It is from this structure that a CDM can determine a device's type and obtain its handle for routing I/O.

Syntax:

```
typedef struct DeviceInfoStruct
    {
        LONG deviceHandle;
        BYTE deviceType;
        BYTE initNumber;
        BYTE busID;
        BYTE cardNo;
        LONG attributeFlags;
        LONG maxDataPerTransfer;
        LONG maxLengthSGElement;
        BYTE maxSGElements;
        BYTE reserved1[2];
        BYTE elevatorThreshold;
        LONG maxUnitsPerTransfer;
        WORD haType;
        union /* Device specific information */
        {
            struct /*SCSI Synchronous Information */
            {
                BYTE transferPeriodFactor;
                BYTE offset;
            } SCSI;
            struct /* Other Device Information */
            {
                BYTE reserved2[2];
            } OTHER;
        } INFO;
        struct InquiryInfoStruct InquiryInfo;
    }deviceInfoDef;
```

Parameters: *DeviceHandle* This is a 1-LONG field containing a handle to a device. The HAM generates this handle during *HAM_Scan_For_Devices()*. This device handle is the token that HAM uses to identify and route I/O to a device. The CDM must provide this handle in the HACB in order to issue I/O to a target device. Without this handle, the HAM rejects the HACB because it cannot identify the target device.

deviceType This is a 1-BYTE field containing a value representing the type of device that the inquiry data will describe. The NWPA uses the same codes for device types as SCSI. The following is the NWPA list of device types:

00 - Direct access device (hard disk)
01 - Sequential access device (tape)
02 - Printer device
03 - Processor device
04 - Write once device (worm)
05 - CD-ROM device
06 - Scanner device
07 - Optical memory device (MO)
08 - Media changer device
09 - Communication device
-1 - Undefined type of device

unitNumber This is a 1-BYTE field. For SCSI, this field contains the logical unit number (LUN) of the device. For IDE\ATA, this field indicates the number (0x00 = Master or 0x01 = Slave) of the device.

Note: The NWPA treats the value in this field as a BYTE value.

busID This is a 1-BYTE field. For SCSI, this field contains the device's SCSI ID. For IDE\ATA, this field contains a HAM-generated index that associates the IDE\ATA-controller channel (primary, secondary, tertiary, or quaternary) to the device.

cardNo This is a 1-BYTE field containing the host adapter card number generated by the HAM.

attributeFlags This is a 1-LONG field indicating the attributes associated with a device and the adapter to which it is attached. The following table describes each attribute and shows the bit that enables it:

Flag Bit (MSB) b31... (LSB) b0	Description
0x00000001	Bit 0 is the **Max_Data_Per_Transfer_Flag**. When set, it indicates that the adapter has a maximum number of bytes it can transfer per I/O request. The value for this maximum is found in the **MaxDataPerTransfer** field. When cleared, it indicates that the adapter can handle any transfer size the bus protocol can support.
0x00000002	Bit 1 is the **Elevator_Off_Flag**. When set, it disables automatic sorting of requests in the NWPA's elevator filter. This task is then left either for the HAM/adapter, or it does not happen at all. Note: If the HAM chooses to turn off the elevator by setting this flag, chances for scatter/gather will be almost nil. The NWPA's scatter/gather filter groups requests while they are in the elevator. Disabling the NWPA's elevator will drastically decrease performance.
0x0000004	Bit 2 is the **Scatter_Gather_Flag**. When set, it indicates that the HAM/adapter supports scatter/gather requests. Then, if the **Elevator_Off_Flag** is cleared, the NWPA scatter/gather filter will seek opportunities to build scatter/gather requests. When cleared it indicates that the HAM/adapter does not support scatter/gather, and the NWPA will guarantee that the associated device's CDM-HAM I/O channel will not receive any scatter/gather requests.
0x00000008	Bit 3 is the **Boot_Device_Flag**. When set, it indicates that this device is the boot device. If the HAM can determine the boot device, it has the option to set this bit. If the HAM cannot make the determination, this flag should be cleared. This flag only applies to RISC architectures and is machine specific. Even then, it only applies in cases where knowing the boot device is necessary.

Flag Bit (MSB) b31... (LSB) b0	Description
0x00000010	Bit 4 is the **Below_16MB_Flag**. When set, it indicates that the adapter is limited to only 16MB of address space. When cleared, it indicates that the adapter is not limited to16MB of address space.
0x00000020	Bit 5 is the **Scatter_Gather_Granularity_Flag**. When set, it indicates that the adapter's transfer granularity per scatter/gather element is at byte resolution. When cleared it indicates that the adapter's transfer granularity per scatter/gather element is at sector resolution.
0x00000040	Bit 6 is the **Auto_Error_Sense_Flag**. When set, it indicates that auto error sense is active for the corresponding device. When cleared, it indicates that auto error sense is inactive.
0x00000080	Bit 7 is the **Private_Public_Flag**. When set, it indicates the corresponding device is private. When cleared, it indicates the corresponding device is public.
0x00000100	Bit 8 is the **Hardware_Verify_Flag**. When set, it indicates that the corresponding device can do hardware verifies on write commands. When cleared, it indicates that the corresponding device does not support hardware verifies on write commands. Note: The setting of this bit is the responsibility of the CDM.
0x00000200	Bit 9 is the **Max_Units_Per_Transfer_Flag**. When set, it indicates that the adapter has a maximum number of units it can transfer per I/O request. The value for this maximum is found in the **MaxUnitsPerTransfer** field. When cleared, it indicates that the adapter can handle any unit transfer amount the bus protocol can support.
0x00000400	Bit 10 is the **Elevator_Threshold_Flag**. When set, it indicates that the **ElevatorThreshold** field is valid. When cleared, it indicates that the **ElevatorThreshold** field is not valid.
b11 ... b31	Bits 11 through 31 (MSB) are reserved.

Flag Bit (MSB) b31... (LSB) b0	Description
DEFAULT=0x00000000	Zero is the default value for this field.

maxDataPerTransfer This is a 1-LONG field indicating the maximum number of bytes that the adapter can transfer per I/O request. If a transfer size limit exists for the adapter, the HAM must place the byte limit in this field and set the **Max_Data_Per_Transfer_Flag**. If the adapter can handle any transfer size the bus protocol supports, the HAM should set this field to zero and clear the **Max_Data_Per_Transfer_Flag**.

maxLengthSGElement This is a 1-LONG field where the HAM indicates the maximum size, in bytes, of a single scatter gather element supported by the adapter for the target device.

maxSGElements This is a 1-BYTE field containing a value corresponding to the maximum number of scatter/gather elements the adapter can handle per request for the target device.

reserved1 This is a 2-BYTE field reserved by the NWPA.

elevatorThreshold This is a 1-BYTE field that indicates the minimum number of requests the HAM prefers to be processing at a given time. The **Elevator_Threshold_Flag** must be set to indicate the validity of this field. If the **Elevator_Threshold_Flag** is cleared, any value in this field should be ignored.

maxUnitsPerTransfer This is a 1-LONG field indicating the maximum number of units (i.e. sectors) that the adapter can transfer per I/O request. If a unit transfer limit exists for the adapter, the HAM must place the unit limit in this field and set the **Max_Units_Per_Transfer_Flag**. If the adapter can handle any unit transfer amount the bus protocol supports, the HAM should set this field to zero and clear the **Max_Units_Per_Transfer_Flag**.

haType A 1-WORD field to contain a value representing the adapter type this HAM supports. The following is a list of possible values:

Field Value	Description
1	HAM supports SCSI adapters.
2	HAM supports IDE\ATA adapters
3	HAM supports custom adapters.

INFO.SCSI.transferPeriodFactor This is a 1 BYTE field that reports the synchronous transfer period, which is the minimum time allowed between leading edges of successive REQ pulses and of successive ACK pulses. (This field applies to SCSI devices only and is not used for other device types.)

INFO.SCSI.offset This is a 1 BYTE field that is the maximum number of REQ pulses allowed to be outstanding before the leading edge or its corresponding ACK pulse is received at the target. Defined values for this field are:

00h	=	Asynchronous transfer
FFh	=	Infinite (No limit to the number of outstanding pulses, which means that memory is fast enough to keep up with synchronous transfer).

(This field applies to SCSI devices only and is not used for other device types.)

INFO.OTHER.reserved2 This is a 2 BYTE field that is reserved by the NWPA. (This field applies to all non SCSI devices.)

inquiryInfo This is a 36 byte (SCSI) / 512 byte (IDE\ATA) field containing an **InquiryInfoStruct** with identifying information about the device. For SCSI, the information in the **InquiryInfoStruct** is identical to the information returned by the standard INQUIRY command. For IDE\ATA, the information in the **InquiryInfoStruct** is identical to the information returned by the IDENTIFY command. For other interface types, the **InquiryInfoStruct** must be defined to contain information identical to the data returned by interface's equivalent INQUIRY command.

ErrorSenseInfoStruct

Used by: CDM and HAM

Description: This structure defines the data format of the HACB's auto error sense buffer.

The CDM allocates and fills in one of these buffers for each HACB request targeted to a device attached to an adapter using auto error sense. The CDM links one of these buffers to a HACB by assigning the buffer's NetWare logical (virtual) address to the HACB's **vErrorSenseBufferPtr** field. The CDM may want to create a reusable pool of these buffers for the sake of performance. Additionally, the buffer must be allocated as I/O contiguous memory, and as explained under the structure's **ErrorSenseData** field presented below, the CDM can vary the size of this buffer according to the number of sense bytes it wants returned. The CDM specifies this number in the **numberBytesRequested** field.

The HAM copies the auto error sense data into the **ErrorSenseData** field of this buffer. Also, the HAM returns to the **numberBytesReturned** field, the lesser of the value in the **numberBytesRequested** field or the actual number of sense bytes the device will provide.

If the number of sense bytes returned by the device is less than what the CDM requested (**numberBytesReturned** < **numberBytesRequested**), the CDM should use the value in the **numberBytesReturned** field as the index for the **ErrorSenseData** array.

Syntax:

```
struct ErrorSenseInfoStruct
   {
      LONG numberBytesRequested;
      LONG numberBytesReturned;
      LONG reserved[2];
      BYTE errorSenseData[1];
   };
```

Parameters: *numberBytesRequested* This is a 1-LONG field to contain the number of error sense bytes the CDM issuing the HACB would like to receive when an error with a check condition occurs. When auto error sense is active for a target device, the CDM assigns the desired value in this field prior to executing the HACB request.

> **Note:** For SCSI, the minimum value a CDM can place in this field is 8. Otherwise, no error sense information will be returned.

numberBytesReturned This is a 1-LONG field to contain the number of error sense bytes that the device actually returned, if the number is less than the number the CDM requested. The HAM sets this value when a HACB

request results in an error with a check condition and the target host adapter has auto error sense turned on. The HAM should set this field according to the following formula:

numberBytesReturned = min(numberBytesRequested, *bytesReturnedByDevice*);

The following assumptions apply to the above formula:

- The CDM must be informed when the length of the sense information returned by the device is less than what the CDM requests.

- The CDM is not concerned with any additional sense information beyond the amount it requested.

reserved This is a field of 2-LONGs reserved by the NWPA.

errorSenseData This field is declared as a BYTE array with one element. The NWPA, however, takes advantage of the fact that the C programming language does not bounds check the array. Therefore, the array's base address (`&ErrorSenseData[0]`) is used as the starting address where the HAM is to place the target device's auto error sense data.

The CDM decides the actual size of this BYTE array, at run-time, when it allocates the auto error sense buffer during the building of the HACB. To get an auto error sense buffer of suitable size, the CDM allocates a buffer the size of the **ErrorSenseInfoStruct** plus however many BYTES of auto error sense data it wants returned. This amount is the value that the CDM assigns to the **numberBytesRequested** field; thus, this field specifies the array's total number of elements.

> **Note:** In building a HACB for a target device with auto error sense active, the CDM assigns the total byte length (sizeof(struct ErrorSenseInfoStruct) + numberBytesRequested) of the auto error sense buffer to the HACB's *errorSenseBufferLength* field.

The CDM and HAM should go through a pointer to an *ErrorSenseInfoStruct* to access information in the auto error sense buffer. This pointer implies an *ErrorSenseInfoStruc*t format on the buffer's data, allowing the CDM or HAM to correctly dereference its fields. The HAM knows the full size of the buffer from the value the CDM places in the *numberBytesRequested* field and adding the 17 header BYTES. The CDM knows exactly how much return data to read by the value the HAM places in the *numberBytesReturned* field.

HACBStruct

Used by: CDM, HAM and NWPA

Description: The Host Adapter Control Block (HACB or **HACBStruct**) is a data structure, or message packet, packing I/O requests into a protocol-specific command block (such as SCSI or IDE\ATA). This structure is passed between a Custom Device Module (CDM) and a Host Adapter Module (HAM) via the NWPA. These modules interface with the NWPA through the CDI and HAI interfaces, respectively.

The HACB is encapsulated in the Super Host Adapter Control Block (SuperHACB or **SuperHACBStruct**), which is a data structure providing additional space for CDM developers to attach additional CDM state information. The CDM uses a SuperHACB to build a device-specific I/O request from a CDM message (**CDMMessageStruct**) it receives from the NWPA. As a data member of the SuperHACB, the CDM places device specific commands in the HACB and initiates its execution by sending it to the HAM via the NWPA. The HAM passes the information in the HACB to the target device for processing.

Syntax:

```
typedef struct HACBStruct
{   LONG hacbPutHandle;
    LONG hacbCompletion;
    LONG control_Info;
    WORD hacbType;
    WORD timeoutAmount;
    LONG deviceHandle;
    LONG dataBufferLength;
    void *vDataBufferPtr;
    void *pDataBufferPtr;
    LONG errorSenseBufferLength;
    void *vErrorSenseBufferPtr;
    void *pErrorSenseBufferPtr;
    LONG reserved1[6];
    BYTE hamReserved[64];
    union /* - - - Command Block Overlay Area - - - */
    {
     struct /* HACB Type 0:Host Adapter Command Structure*/
     {
        LONG function;
        LONG parameter0;
        LONG parameter1;
        LONG parameter2;
        BYTE reserved2[12];
     } Host;
     struct /* HACB Type 1: SCSI Adapter Command Structure*/
     {
        BYTE haCommandArea[16];
        BYTE reserved3[11];
        BYTE haCommandLength;
     } SCSI;
     struct /* HACB Type 2: IDE\ATA Adapter Command
Structure*/
     {
        BYTE numberSectorsRegister;
        BYTE sectorRegister;
```

```
            BYTE lowCylinderRegister;
            BYTE highCylinderRegister;
            BYTE driveHeadRegister;
            BYTE commandRegister;
            BYTE reserved4[22];
          } IDE\ATA;
          struct /*HACB Type 3:CDM Pass-through Cmd Structure*/
          {
            LONG function;
            LONG parameter0;
            LONG parameter1;
            LONG parameter2;
            BYTE reserved5[12];
          } CDMPassThrough;
        } Command;
} HACB;
```

Parameters: A full description of the **HACBStruct** parameters is not given here due to its length and detail. Refer to Chapter 3 for a full description.

HAMInfoStruct

Used by: HAM

Description: This structure is used by a HAM to supply information about the HAM itself to the Media Manager upon request. The HAM needs to maintain an instance of this structure for each bus it supports.

Syntax:

```
struct HAMInfoStruct
{
    LONG deviceInfoStructureLength;
    WORD haType;
    BYTE busNo;
    BYTE cardNo;
    LONG vendorID;
    BYTE name[64];
    LONG supportedTargetIDs;
    LONG supportedUnitNumbers;
    LONG cardTargetID;
    LONG reserved[10];
};
```

Parameters: *deviceInfoStructureLength* A 1-LONG field to contain the length of the device information data. For SCSI devices, this value is the length of the header (32 bytes) plus the SCSI Inquiry Data (36 bytes). For IDE\ATA devices, this value is the length of the header (32 bytes) plus the IDE\ATA Information (512 bytes). For custom CDMs and HAMs, this value is the length of the header (8 bytes) plus the length of the custom information.

haType A 1 WORD field to contain a value representing the adapter type this HAM supports. The following is a list of possible values:

Field Value	Description
1	HAM supports SCSI adapters.
2	HAM supports IDE\ATA adapters
3	HAM can translate raw Media Manager messages into custom command blocks for the adapter it supports.

busNo A 1 byte field to contain the numerical identifier used by the HAM to indicate the appropriate bus on which to process a HACB. This identifier accommodates those adapters that have more than one bus on which to attach devices. This number is set by the HAM.

cardNo A 1 byte field to contain the number that will be displayed for this adapter and used to identify the adapter in other commands. This number is decided by the HAM.

vendorID A 4 byte field to contain a number used to keep track of all

modules. This number is given to a driver vendor from Novell Labs and should be hard-coded in the module. This number is used in registering a module and in hot replacement.

name[64] A 64 byte field to contain the name of the adapter or the HAM. The name is a string where byte 0 contains the string length and bytes 1 through 63 contain the characters that constitute the actual name.

supportedTargetIDs A 1 LONG field to contain the number of Target IDs supported by this HAM. This corresponds to ID numbers in the case of SCSI, and Channel numbers in the case of IDE/ATA.

supportedUnitNumbers A 1 LONG field to contain the number of Unit Numbers supported by this HAM. This corresponds to LUNs in the case of SCSI.

cardTargetID A 1 LONG field to contain the specific card ID that this HAM will support, if known by the HAM. If this parameter is not used, it must be set to -1.

reserved These 10 LONGs are reserved by NWPA.

InquiryInfoStruct

Used by: CDM and HAM

Description: This structure contains identifying information that the CDM and HAM can use to know what type of device is being described. For SCSI, **InquiryInfoStruct** is identical to the SCSI Inquiry structure returned by the SCSI Inquiry Command. For IDE\ATA, **InquiryInfoStruct** is identical to the IDE\ATA Identify structure returned by the IDE\ATA Identify Command. For other interface types, **InquiryInfoStruct** must be identical to the data structure specific to that interface type.

NPAOptionStruct

Used by: HAM (CDM usage is optional)

Description: The **NPAOptionStruct** contains the HAM's command line option data on a per option basis. Using this structure, the HAM can select the command line options that it wants the Media Manager to prompt the system operator for. The HAM must fill out one of these structures and call **NPA_Add_Option()** for each option it supports. With each successive call to **NPA_Add_Option()**, the Media Manager adds the current option to a select list. After the HAM has added all of its command line options, it calls **NPA_Parse_Options()**, which parses the command line to determine which options in the select list were actually chosen. Within the context of **NPA_Parse_Options()**, the Media Manager iteratively calls the HAM's *HAM_Check_Option()* routine for each option that was actually selected from the command line. *HAM_Check_Option()* can direct the Media Manager to either accept the option by returning zero or reject the option by returning non-zero. If the option is accepted, the Media Manager places it in a use list. The HAM then calls **NPA_Register_Options()** to direct the Media Manager to physcially register the options in its use list for the HAM.

The Media Manager will not place multiple options of the same type, such as multiple interrupts, in its use list for a single parse of the command line. Therefore, if the host adapter supports multiple options of the same type and the HAM wants to exploit them, then the HAM must do the following:

1. Call **NPA_Add_Option()** to add the first option.
2. Call **NPA_Parse_Options()** and have *HAM_Check_Option()* accept the option so that it is placed in the use list.
3. Call **NPA_Add_Option()** to add the next option of the same type.
4. Call **NPA_Parse_Options()** and have *HAM_Check_Option()* accept this option so that it is also placed in the use list.
5. Repeat steps 3 and 4 until all of the options of the same type are in the use list.
6. Call **NPA_Register_Options()** to have the Media Manager physically register the options.

Syntax:

```
struct NPAOptionStruct{
    BYTE name[32];
    LONG parameter0;
    LONG parameter1;
    LONG parameter2;
    WORD type;
    WORD flags;
    BYTE string[n];
} ;
```

Parameters: *name* This is a 32 byte field to contain a length-preceded and null-terminated string. The HAM places the name of the desired option, as it

will appear on the command line, in this field.

> **Note:** NWPA will add an equals sign after *Name* when it is displayed on the command line.

parameter0 This is a 4 byte field to contain the value associated with an option.
For the interrupt option, this field would contain the IRQ level.

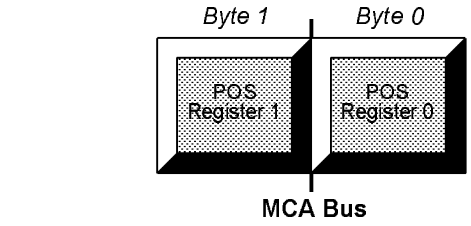
For the memory decode option where an adapter card has onboard memory that must be mapped into NetWare's logical address table, this field would contain the shared memory absolute address used by the adapter.

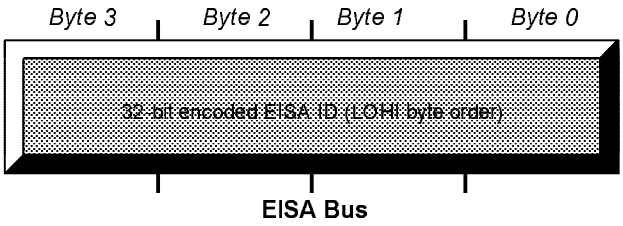
parameter1 This is a 4 byte field to contain the length or range associated with this option. Typically, this field is used in specifying memory decode ranges and port lengths.

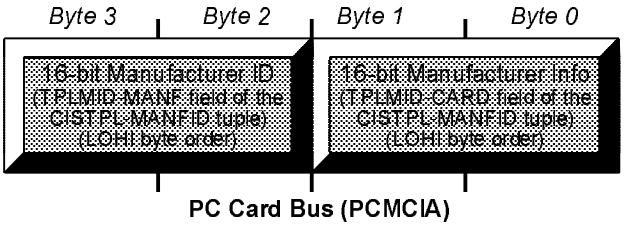
If the Interrupt Option is set under *Type*, this field represents the following flags:

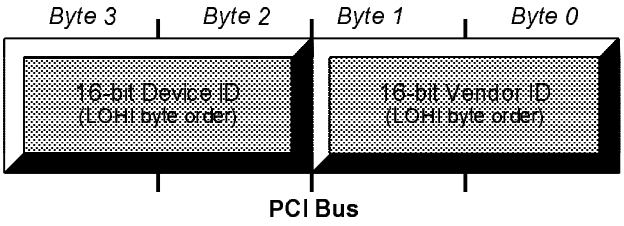
- 0x01 - Put at end of ISR chain (Default is front of ISR chain.)
- 0x02- Adjust RealModeInterrupt mask. This enables real mode (DOS) Interrupts.
- 0x04- Level triggered Interrupt (Default setting is edge-triggered.)

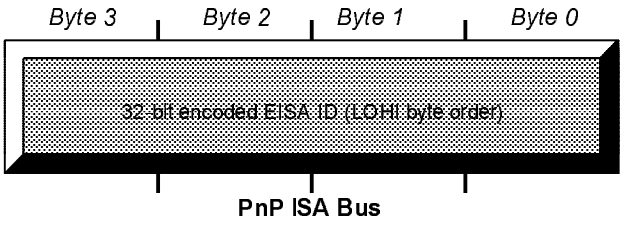
For the Product ID option, this parameter contains a pointer to an array of bytes that contain a bus architecture-dependant parameter that uniquely identifies an adapter board/peripheral/system option. As an example, in the case of an EISA bus, the EISA product ID is defined in the EISA Specification document. The following illustration shows the various formats for product ID values (as applicable).











Note: LOHI byte order refers to a little-endian byte order.

parameter2 This is a 4 byte field that can be either an input or an output parameter. In the shared memory case mentioned previously, this field receives the logical address of the mapped memory. For Interrupt, Port,

Memory, Slot, and DMA options, this parameter is the *busTag* as defined for NBI that is returned by **NPAB_Get_Bus_Tag()**. For the Product ID option, this field is the size of the array pointed to by **Parameter1**.

> **Note:** Return values to this parameter are only valid after **NPA_Register_Options()** has been called.

type This is a 2 byte field to contain a code indicating the option type.

The following is a list of possible values for this field:

0x0000 HAM-defined option (such as debug)
0x0001 Interrupt option
0x0002 Port option
0x0003 DMA option
0x0004 Memory decode option
0x0005 Slot option
0x0006 Card option
0x0007 Reserved by NetWare
0x0008 Product ID option
0x0009
to Reserved by NetWare
0x00FF
0x0100
to For Vendor use as needed.
0xFFFF

flags This is a 2 byte field to contain a bitmap indicating the status of the option. The following is a list of the flags defined for this field.

0x0001 Option required -- If not specified on command line, then prompt the user.
0x0002 Use this option -- Use this option whether or not it is specified on the command line.
0x0004 Value required -- Places *name* = on the command line where *name* is the string contained in the *Name* field and the user is expected to enter a value.
0x0008 Specific value required -- Places *name* = and a set of specific values on the command line from which the user is expected to choose one. Each value in the set is contained in **parameter0** of the option's corresponding **NPAOptionStruct**.
0x0010 Default value -- Contained in **parameter0**
0x0020 Shareable option -- Such as shared interrupts
All other bits in this field are reserved by NetWare.

string This is a *n*-byte field that can be used to pass and\or receive information to/from the command line. If the Specific Value Required flag is set, this field will contain a length-preceded and NULL terminated string where *n* is an arbitrary integer (determined by the HAM developer) that is a multiple of 4 (LONG aligned). This field contains the ASCII code for the value specified in **parameter0**. In this case where a matching option was not specified on the command line, this value appears at the console as a choice for the user. After a user makes a selection, the selected value is placed back into this field.

If the developer desires to use this field to return information back from the command line, (Value Required flag is set) this field must contain *n*-2, where *n* is the maximum length of *String* plus the length count byte and the NULL terminator byte. In this case, when the information is returned back, the length byte will be updated to indicate the actual size of the string being returned.

SuperHACBStruct

Used by: CDM

Description: The Super Host Adapter Control Block (**SuperHACB** or **SuperHACBStruct**) is a data structure, or message packet, packing I/O requests into a protocol-specific command block (such as SCSI or IDE\ATA). It provides additional space for CDM developers to attach additional CDM state information, and it encapsulates a Host Adapter Control Block (**HACB** or **HACBStruct**) which is the structure passed between a Custom Device Module (CDM) and a Host Adapter Module (HAM) via the Media Manager. The CDM uses a **SuperHACB** to build a device-specific I/O request from a CDM message (**CDMMessageStruct**) it receives from the Media Manager. As a data member of the **SuperHACB**, the CDM places device specific commands in the **HACB** and initiates its execution by sending it to the HAM via the Media Manager. The HAM passes the information in the **HACB** to the target device for processing.

Syntax:

```
typedef struct SHACBStruct
{
    LONG cdmSpace[8];
    struct HACBStruct HACB;
} SHACB;
```

Parameters: *cdmSpace* This is a 32 -byte field to be used at the CDM's discretion. This field may be used to store state information specific to a CDM, but the use of this field is optional. However, if this field is used, the CDM is responsible for setting its values.

HACBStruct HACB This is a field containing a HACB structure defined in section 3.3. A SuperHACB structure pointer is what the Media Manager APIs pass to and from a CDM. The HAM only receives and acts on the information contained in the HACB structure.

UpdateInfoStruct

Used by: CDM

Description: This structure is used by a CDM when binding to a device or when updating device information. Most importantly, the CDM uses this structure to register the control and I/O functions it will support for a device with the Media Manager.

Syntax:

```
struct UpdateInfoStruct
{
    BYTE name[64];
    LONG mediaType;
    LONG cartridgeType;
    LONG unitSize;
    LONG blockSize;
    LONG capacity;
    LONG preferredUnitsize;
    LONG functionMask;
    LONG controlMask;
    LONG unfunctionMask;
    LONG uncontrolMask;
    LONG mediaSlot;
    BYTE activateFlag;
    BYTE removableFlag;
    BYTE readOnlyFlag;
    BYTE magazineLoadedFlag;
    BYTE acceptsMagazinesFlag;
    BYTE objectInChangerFlag;
    BYTE objectIsLoadableFlag;
    BYTE lockFlag;
    LONG diskGeometry;
    LONG reserved[7];
    union
    {
        struct ChangerInfo
        {
            LONG numberOfSlots;
            LONG numberOfExchangeSlots;
            LONG numberOfDevices;
            LONG deviceObjects[n];
        } ci;
    } u1;
} ;
```

Parameters: *name* This field is a length-preceded string to contain the manufacturer's name and model number of the device.

mediaType This is the type of media being used.

disk	0x00000000
tape	0x00000001
printer	0x00000002
WORM	0x00000004
CDROM	0x00000005
magneto optical	0x00000007

cartridgeType The type of any cartridge if the device supports one

fixed media	0x00000000
5.25 floppy	0x00000001
3.5 floppy	0x00000002
5.25 optical	0x00000003
3.5 optical	0x00000004
.5 tape	0x00000005
.25 tape	0x00000006
8 mm tape	0x00000007
4 mm tape	0x00000008
Bernoulli disk	0x00000009

unitSize The current transfer unitsize (bytes per sector) setting of the device. This is the transfer unitsize in which the base-translator CDM will receive requests.

For Disk, CD-ROM, and MO devices, this field should contain the unitsize native to the media in the device. This is the unitsize that either optimizes device performance or is physically imposed on the device by the media, as in the case of CD-ROM. If the value in this field is anything other than 512 (NetWare's native unitsize), the NWPA's sector translation filter gets turned on to ensure that the CDM will receive requests in the unitsize specified by this field.

For Tape devices, the CDM should never change the value in this field unless an application tells it to physically change the device's unitsize through its *CDM_Set_Attribute()* routine. Then, and only then, will the CDM place the new unitsize value in this field and update the object using **CDI_Object_Update()**. It is the responsibility of the application using the tape device to issue requests in the unitsize specified by this field.

blockSize Indicates the maximum number of transfer units that can be specified in a single command (i.e. sectors per request). The NWPA uses this value to make sure that the CDM does not receive blocks that are too big for it to handle. The CDM should set the blocksize to the smaller of either the maximum number of transfer units the CDM can handle per request or the maximum blocksize imposed by the adapter. The CDM is informed that an adapter blocksize limitation exists if either the **Max_Data_Per_Transfer_Flag** (0x00000001) or the **Max_Units_Per_Transfer_Flag** (0x00000200) is set in the **attributeFlags** field of the device's **DeviceInfoStruct**. The CDM receives a pointer to the device's **DeviceInfoStruct** as an input parameter to its *CDM_Inquiry()* routine.

The adapter imposed blocksize is determined by the following criteria: 1.

If the **Max_Data_Per_Transfer_Flag** is set and the **Max_Units_Per_Transfer_Flag** is cleared, then the CDM calculates the adapter imposed blocksize by dividing the value in the **MaxDataPerTransfe**r field of the device's **DeviceInfoStruct** by the value in the **Unitsize** field of this structure (the device's **UpdateInfoStruct**).

2. If the **Max_Units_Per_Transfer_Flag** is set and the **Max_Data_Per_Transfer_Flag** is cleared, then the CDM uses the value in the **MaxUnitsPerTransfer** field of the device's **DeviceInfoStruct** as the adapter imposed blocksize.
3. If both flags are set, then the CDM uses the smaller of 1 or 2 above as the adapter imposed blocksize.

capacity The capacity of the media in the device in terms of transfer units (i.e. total number of sectors). For those types of media, such as tape, where capacity of the media is not readily available, it is preferrable that the CDM approximate the capacity. However, if approximating the capacity is too difficult, the CDM should set this field to -2, which indicates capacity unknown.

preferredUnitsize The transfer unitsize (bytes per sector) in which the base-translator CDM would prefer to receive requests. For Disk, CD-ROM, and MO devices, the value in this field should be equal to the value specified in the **Unitsize** field. This way, the NWPA's sector translation filter ensures that the base-translator CDM receives requests in the unitsize it specified in the Unitsize field. For Tape devices, the value in this field is a hint to tape applications of the preferred transfer unitsize. Applications can choose to use this hint or ignore it.

functionMask A 32-bit mask indicating the I/O functions the CDM will support for this device. The CDM may update this field as needed.

```
#define RANDOM_READ            0x00000001
#define RANDOM_WRITE           0x00000002
#define RANDOM_WRITE_ONCE      0x00000004
#define SEQUENTIAL_READ        0x00000008
#define SEQUENTIAL_WRITE       0x00000010
#define RESET_END_OF_MEDIA 0x00000020
#define SINGLE_FILE_MARKS      0x00000040
#define MULTIPLE_FILE_MARKS    0x00000080
#define SINGLE_SET_MARKS       0x00000100
#define MULTIPLE_SET_MARKS 0x00000200
#define SPACE_DATA_BLOCKS      0x00000400
#define LOCATE_DATA_BLOCKS 0x00000800
#define PARTITION_SUPPORT      0x00001000
#define SEQUENTIAL_SUPPORT 0x00002000
#define MO_ERASE           0x00004000
#define VENDOR_UNIQUE_IO       0x40000000
```

controlMask A 32-bit mask indicating the control functions the CDM will support for this device. The CDM may update this field as needed.

```
#define FORMAT_MEDIA                0x00000001
#define TAPE_CONTROL                0x00000002
#define ACTIVATE_DEACTIVATE_MASK    0x00000008
#define MOUNT_DISMOUNT_MASK         0x00000010
#define SELECT_DESELECT_MASK        0x00000020
#define LOAD_UNLOAD_MASK            0x00000040
#define LOCK_UNLOCK_MASK            0x00000080
#define MOVE_MEDIA_MASK             0x00000100
#define LOAD_MAGAZINE_MASK        0x00002000
#define CHANGER_INVENTORY_MASK      0x00004000
#define RAW_INSERT_MASK             0x08000000
#define RAW_CHANGER_MASK            0x10000000
#define RAW_MAGAZINE_MASK           0x20000000
#define VENDOR_UNIQUE_CONTROL       0x40000000
```

unfunctionMask This field is used by filter CDMs. Its value is a 32-bit mask that has bits set for each function that is to be removed from the current function mask.

uncontrolMask This field is used by filter CDMs. Its value is a 32-bit mask that has bits set for each control function that is to be removed from the current control mask

mediaSlot This field is reserved by the NWPA.

activateFlag Set to 1 if the device is active or 0 if the device is inactive.

removableFlag Set to 1 if the device holds removable media. Set to 0 if the device holds non-removable media (i.e. fixed disks).

readOnlyFlag Set to 1 if the media in the device is read-only or write-protected media. Set to 0 if the media in the device is readable and writable.

magazineLoadedFlag Set to 1 if the device has a magazine currently loaded. Set to 0 if the device does not have a magazine currently loaded. Set to -1 if the device does not support magazines.

acceptsMagazinesFlag Set to 1 if the device supports magazines. Set to -1 if the device does not support magazines.

objectInChangerFlag Set to 1 if this device is located inside a changer. Set to 0 if the device is not inside a changer.

objectIsLoadableFlag Should be set if the object can be loaded.

lockFlag Set to 1 if the device has locked the removable media in its drive slot. Set to 0 if the removable media is not locked. Set to -1 if the device does not support Prevent/Allow Medium Removal commands.

diskGeometry Indicates the disk geometry if the device is a hard disk that does not support logical block addressing (LBA). If the device does support LBA, then set this field to -1. The value in this field is treated as a LONG (32-bits). The value in bits (LSB) 0 - 7 indicates the sectors per track. The value in bits 8 - 15 indicates the number of heads. The value in bits 16 - 31 (MSB) indicates the number of cylinders.

reserved Reserved by NetWare.

changerInfo

numberOfSlots Used to set the number of slots in an autochanger.

numberOfExchangeSlots Used to set the number of mailboxes in an autochanger.

numberOfDevices Used to set the number of devices in an autochanger.

deviceObjects A list of the device. n is an arbitrary integer chosen by the CDM developer.