Chapter 3 Host Adapter Control Block (HACB)

This chapter describes the SHACB and HACB data structures. The chapter begins with an overview that describes each structure's role and function within the NWPA. The SHACB is described first, then the HACB. Each description includes a prototype of the structure and descriptions of its fields.

3.1 Overview

The Host Adapter Control Block (HACB) is a data structure, or information packet, passed between a Custom Device Module (CDM) and a Host Adapter Module (HAM). The Super Host Adapter Control Block (SHACB) is a data structure that envelopes a HACB allowing CDM developers to attach additional CDM state information, if needed. However, the HAM uses only the information in the HACB to process requests. CDMs allocate SHACBs via a call to **CDI_Allocate_HACB()**, and they are returned via a call to **CDI_Return_HACB()**. CDMs and HAMs do not interface directly; therefore, HACBs are not passed directly between them. The NWPA is the communication channel, and it passes HACBs between CDMs and HAMs. This flow path is depicted by the diagrams in section 2.2. The NWPA interfaces with each module through the entry points specified at each module's registration. CDMs interface with the NWPA through the **CDI_** API set, and HAMs interface with the NWPA through the **HAI_** API set. The NWPA, CDM, and HAM are responsible for setting certain field values in a HACB at different stages of execution. The field descriptions identify the component responsible for setting a field value. Note: A SHACB allocated with **CDI_Allocate_HACB()** is not guaranteed to be below the 16 megabyte boundary.

3.2 Super Host Adapter Control Block (SHACB)

Each HACB created with **CDI_Allocate_HACB()** is enveloped in a data structure called a SHACB. This section defines the SHACB structure and gives a description of its fields.

3.2.1 Structure Definition

The SHACB definition is as follows:

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

3.2.2 Field Descriptions

This section describes each field within a SHACB structure:

CDMSpace
This is an 8-LONG 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.

HACB
This is a field containing the HACB structure defined in section 3.3.

3.3 HACB Structure

The HACB structure is a data packet containing a device control or I/O command issued by a CDM or the NWPA. This section defines the HACB structure and gives a description of its fields. Note: Certain fields in the HACB are pre-initialized by the NWPA at allocation, and their values must be maintained. Therefore, do not clear or zero out the HACB. In the SFT III (System Fault Tolerance) environment, only the data in the HACB's data buffer, error sense buffer, **hacbCompletion**, and **Control_Info** fields get mirrored between the fault tolerant servers.

3.3.1 Structure Definition

The following is the ANSI C definition of the **HACBStruct** (HACB):

```
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 Cmd Structure*/
        {
            LONG function;
            LONG parameter0;
            LONG parameter1;
            LONG parameter2;
            BYTE reserved2[12];
        } Host;
```

```
        struct /*HACB Type 1:SCSI Adapter Cmd Structure*/
        {
            BYTE haCommandArea[16];
            BYTE reserved3[11];
            BYTE haCommandLength;
        } SCSI;
        struct /*HACB Type 2:IDE\ATA Adapter Cmd 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;
```

3.3.2 Field Descriptions

The subsections that follow describe each field within the HACB structure.

hacbPutHandle

This is a 1-LONG field containing a unique handle identifying the current SHACB or HACB. When passing HACB requests, CDMs and HAMs use this handle instead of passing memory pointers.

Note: CDMs and HAMs must not alter this handle. Also, do not confuse this field with the **MsgPutHandle** field of the **CDMMessageStruct**. Their values and purposes are different. The NWPA uses this handle to track a HACB through different execution stages. Many of the APIs described in this manual, such as **HAI_Complete_HACB()**, need this handle as an argument. When a SHACB is first allocated using **CDI_Allocate_HACB()**, the NWPA initializes this field of the enveloped HACB with a value. When the HACB is sent for execution (this occurs when a CDM calls either **CDI_Execute_HACB()** or **CDI_Blocking_Execute_HACB()** on the HACB) the NWPA generates and places a new value in this field handle.

hacbCompletion

This is a 1-LONG field containing the completion status (successful or unsuccessful) of a HACB request. The NWPA allows for status codes under the following interfaces: SCSI, IDE/ATA, and Custom. This field is set by the HAM. The HAM completes a HACB request by calling **HAI_Complete_HACB()**, which then informs the CDM layer. The HAM routine handling HACB completion, however, must post status to this field prior to calling **HAI_Complete_HACB()**. The CDM reads the value in this

field during its callback entry point, *CDM_Callback()*, to determine whether the HACB completed successfully or not, and to determine the target device's current queue state. A hierarchy is associated with the status values placed in this field. The upper WORD (16 bits) indicates the following:

- The current queue state of the device that processed the HACB request. The most-significant-bit (**MSBit**) is the post-completion queue state indicator. If the MSBit=1 at CDM callback-time, then the CDM knows that the HAM froze the device queue after processing this HACB, thereby suspending the processing of all other HACBs positioned after it. This condition indicates that either an error occurred in processing the current HACB, or the HAM was told to freeze the queue following HACB completion by the CDM setting the **Freeze_Queue_Flag** in the **Control_Info** field. If the **MSBit**=0 at CDM callback-time, then the HAM did not freeze the queue after processing the current HACB, and subsequent HACBs are still being processed.

- The general category of the HACB's completion status. The general category is determined by the value of the remaining 15 bits in the upper WORD.

The lower WORD gives further resolution by being a qualifier that indicates additional status information. For some categories, however, the value in the lower WORD is either not applicable or undefined in the NWPA. For processor independence reasons, this field needs to be processed as a LONG. Therefore, HAMs and CDMs must use macros to encode and decode this field. The HAM must define the following macro and use it to encode a HACB's completion status:

```
#define SET_STATUS (UpperWord, LowerWord) ( (UpperWord )<<
16) | ((LowerWord) & 0xFFFF))
```

The CDM must define the following macros and use them in its callback to decode the completion status of the HACB and the device, respectively:

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

Appendix B lists the HACB completion status values currently defined in the NWPA along with a detailed set of descriptions for each value.

control_Info

This is a 1-LONG field that is bi-directional. The CDM uses it to pass control information to the HAM, and the HAM uses it to pass status information back to the CDM. The remaining description of this field is divided according to HACB direction.

CDM to HAM

When the HACB direction is from the CDM to the HAM, it generally

indicates that the CDM has received an I/O message from an upper layer for which it will build a HACB I/O request recognized by the HAM. In this case, this field is to contain a bitmap of operational control flags. These flags are set by the CDM to indicate operational conditions associated with a HACB request. The HAM reads these flags to determine the conditions specified by the CDM. The setting of control flags for a HACB request reflects the most common use of this field. The CDM control or I/O routine that is building the HACB has the flexibility to toggle these flags to set the proper combination needed for the request. Table 3-1 lists the control flag values currently defined in the NWPA along with their respective descriptions.

Table 3-1: HACB Control Flag Values

Flag Value	Description
0x00000001	Bit 0 (LSB) is the **Priority_HACB_Flag**. When set, it indicates to the HAM that this request is a priority request giving it precedence over non-priority requests in the HAM's device process queue. When cleared (zero), it indicates that this request is a non-priority request. Priority requests are ordered on a Last-In-First-Out (LIFO) basis.
0x00000002	Bit 1 is the **Data_Direction_Flag**. When set, data flow is to the device. When cleared, data flow is from the device.
0x00000004	Bit 2 is the **Freeze_Queue_Flag**. The CDM can cause single-step execution of HACB requests by toggling this bit. If the CDM sets this bit in the HACB--and sends the HACB to the HAM for processing--it indicates to the HAM that it must freeze this device's process queue immediately after issuing this HACB request to the device. If the CDM clears the bit (zero), it indicates to the HAM to continue normal operation of the device's process queue even after issuing this HACB to the device. Bit 9 (**No_Freeze_Queue_Flag**) and this bit cannot both be set at the same time. If both bits are set, a Malformed HACB error will result.
0x00000008	Bit 3 is the **Timeout_Granularity_Flag**. When set, the bit indicates that the value specified in the **TimeoutAmount** field is in minutes. When cleared, the bit indicates that the value specified in the **TimeoutAmount** field is in seconds.
0x00000010	Bit 4 is the **Scatter_Gather_Flag**. When set, it indicates the following to the HAM: • The **VDataBufferPtr** field contains the starting virtual address of a scatter/gather request list. • The **PDataBufferPtr** field contains the starting physical address of the same scatter/gather request list. • The **DataBufferLength** field contains the number of entries in the scatter/gather request list. The maximum number of scatter/gather entries allowed for a device is specified in the **MaxSGElements** field of that device's **DeviceInfoStruct**. When cleared, it indicates the following: • The **VDataBufferPtr** field contains the starting virtual address of the data buffer. • The **PDataBufferPtr** field contains the starting physical address of the same data buffer. • The **DataBufferLength** field contains the total data buffer length in bytes.
0x00000020	Currently reserved by NetWare.

Flag Value	Description
0x00000040	Bit 6 is the **Preserve_Order_Flag**. When set, the HAM preserves the current request order in the device queue. When cleared, the requests in the device queue can be ordered as prescribed by the HAM. This feature is provided so that a CDM supporting sequential devices can control the order of request execution.
0x00000080	Bit 7 is the **No_Disconnect_Flag** for use with SCSI. When set, it indicates no disconnect.
0x00000100	Bit 8 is the **No_Data_Transfer_Flag**. When set, it indicates that the issued request does not require the transfer of data, and the function of the **Data_Direction_Flag** is ignored. If the CDM sets this flag it must also zero out the HACB's *DataBufferLength* field. Setting this flag does not affect the HACB's error sense buffer or its length. When cleared, it indicates that the issued request requires the transfer of data. The direction of transfer is indicated by the **Data_Direction_Flag**
0x00000200	Bit 9 is the **No_Freeze_Queue_Flag**. The CDM can prevent the queue from being frozen regardless of error condition by setting this bit. Bit 2 (**Freeze_Queue_Flag**) and this bit cannot both be set at the same time. If both bits are set, a Malformed HACB error will result.
0x00000400 to 0x80000000	Bits 10 through 31 (MSB) are reserved by NetWare.
DEFAULT= 0x00000000	Zero is the default value for this field.

HAM to CDM
When the HACB direction is from the HAM to the CDM, it generally indicates that the device has completed the I/O request, and the HAM is ready to post completion status and send the HACB back to the CDM.

The HAM only places a value in this field for the following completion status:

Malformed Error - Data Overrun/Underrun with ActualTransfer Count Available (0x80030003 / 0x80030004) The HAM is managing an adapter that can provide an actual-data-transferred-count for buffer overrun/underrun conditions. In this case, the HAM should post the appropriate status value to the HACB and place the actual number of bytes that were transferred into this field.

hacbType
This is a 1-WORD field containing a code that defines the HACB request type. This field value is set either by the CDM, through one of its control or I/O routines, or by the NWPA. The CDM or NWPA fills the HACB's command block overlay area with a command structure appropriate to the HACB's type. By checking this field, the HAM can determine what command structure to expect in the HACB. Table 3-2 defines the HACB

request types.

Table 3-2: HACB Type Values

HACBType	Description
0x0000	**HACBType=0** requests contain adapter-specific Host command structures issued by the NWPA or event notification requests issued by the CDM. These requests ask for information about the HAM, the host adapter, or attached devices. As explained in Chapter 4, all HAMs must support **HACBType=0** requests.
0x0001	**HACBType=1** requests contain SCSI command structures built by a CDM that supports SCSI devices for a HAM that supports SCSI type adapters.
0x0002	**HACBType=2** requests contain IDE\ATA command structures built by a CDM that supports IDE\ATA devices for a HAM that supports IDE\ATA type adapters.
0x0003	**HACBType=3** requests contain CDMPassThrough command structures built by a CDM for a HAM that supports raw Media Manager requests.
0x0004 to 0x00FF	This range of HACBType values is reserved by NetWare.
0x0100 to 0xFFFF	This range of **HACBType** values is reserved for custom types. These numbers are coordinated and assigned for vendor use by Novell Labs.

timeoutAmount

This is a 1-WORD field containing the number of seconds or minutes within which a device must finish processing a HACB; otherwise, the HAM will time-out the request. The main purpose of this time limit is to provide a recovery point (see *HAM_Timeout()* in Chapter 7) from a hung device. CDMs are required to set a value in this field for all non-zero type HACBs (**HACBType**=1,2,3) it issues to the HAM. The time-out countdown begins when the HAM issues the request to the device. The time spent in the HAM's device queue is not included in the countdown. However, since the HAM may not have any control over a request after it is issued, time spent queued in hardware caches on the adapter (and/or on the device) is included in the countdown. The CDM sets the value in this field since it has the device-specific intelligence to know how long a request should take to complete. However, since a request may spend additional time queued in caches (particularly a cache on the adapter of which the CDM may not be aware), the CDM should allow leeway in assigning this value. The general rule that a CDM should follow is to set an optimal value that will cause a time-out only if typical process time is grossly exceeded. The CDM indicates the time-out granularity, whether a value is in seconds or minutes, by setting the time-out granularity bit in the HACB's **Control_Info** field. When the bit is set, the granularity is in minutes. When the bit is not set, the granularity is in seconds.

> **Note:** The HAM may receive **HACBType=0** requests with this field set to zero. Generally, these requests are asking the HAM to perform adapter-specific functions for which the NWPA has no way of knowing how much time it should take to complete. An example would be a request for the HAM to scan the host bus for attached devices.
>
> For these **HACBType=0** requests, the HAM must make a reasonable decision as to the amount of time it will allow the request to process before timing it out. The HAM must ensure that these requests never hang.

deviceHandle
This is a 1-LONG field containing a HAM-generated handle to the device that the HACB request will be routed to. This field is set by the CDM I/O routine that builds the HACB request. The HAM generates this handle and reports it to the NWPA. The HAM must be able to locate its devices and their respective queues from this handle. The NWPA then makes this handle available to the CDM. In order for a CDM to channel requests to a target device, it must provide the appropriate HAM-generated device handle in this field of the HACB.

dataBufferLength
This is a 1-LONG field set by the CDM. Its content depends on whether or not the **Scatter_Gather_Flag** in the HACB is set. If the flag is set, this field contains the number of elements in the NWPA-generated scatter/gather request list. If the flag is not set, this field contains the length, in bytes, of the request's data buffer. In either case, the field is set by the CDM I/O routine that builds the HACB. This value is obtained from the **BufferLength** field of the corresponding CDM Message (**CDMMessageStruct**) associated with the HACB. The NWPA passed a pointer to this CDM Message as an input parameter to the CDM's entry point that received the HACB. Refer to section 3.4 for a description of the NWPA's scatter/gather format and how it affects this field.

vDataBufferPtr
This is a 4-byte field of type pointer to void, and its contents will be a virtual pointer to the HACB's data buffer that either receives or contains I/O data for read and write operations, respectively. The NWPA provides this field to support host adapter boards that use programmed I/O. The CDM I/O or Control routine that builds the HACB places the appropriate value in this field, which it obtains from the Buffer field of the CDM Message (**CDMMessageStruct**) associated with the HACB. The NWPA passed a pointer to this CDM Message as an input parameter to the CDM's entry point that received the HACB. The structure of the buffer it points at

depends on whether or not the **Scatter_Gather_Flag** is set. If the flag is set, this field contains the virtual starting address of the scatter/gather request list. The scatter/gather list is either generated by an NWPA filter or a Media Manager application. If the flag is not set, this field contains the virtual address of the request's data buffer. Refer to section 3.4 for a description of the NWPA's scatter/gather format and how it affects this field.

pDataBufferPtr
This is a 4-byte field of type pointer to void. The NWPA calculates the physical (absolute) address of the buffer pointed at by **vDataBufferPtr** and places the address in this field. The NWPA provides the physical address to support adapters that use DMA or bus-mastering. The structure of the buffer it points at depends on whether or not the **Scatter_Gather_Flag** is set. If the flag is set, this field contains the physical starting address of the scatter/gather request list. The scatter/gather list is either generated by an NWPA filter or a Media Manager application. If the flag is not set, this field contains the physical address of the request's data buffer. In either case, calculating this field value is not the concern of the CDM at HACB-build time. However, for safety, the CDM I/O routine building the HACB should initialize the field to zero. After the CDM I/O routine calls **CDI_Execute_HACB**(), the NWPA calculates the physical address and places it in this field before sending the HACB to the HAM. Refer to section 3.4 for a description of the NWPA's scatter/gather format and how it affects this field.

> **Note:** The NWPA guarantees this buffer to be physically contiguous.

errorSenseBufferLength
This is a 1-LONG field set by the CDM I/O or Control routine that builds the HACB. This field's value specifies the size, in bytes, of the error sense buffer pointed at by the **vErrorSenseBufferPtr** and **pErrorSenseBufferPtr** fields. Essentially, the value of this field should be the size of the NWPA's **ErrorSenseInfoStruct** plus any additional error sense bytes the CDM chooses to append to this structure. Refer to the **ErrorSenseInfoStruct** reference information in Chapter 6 for more details about this concept.

vErrorSenseBufferPtr
This is a 4-byte field of type pointer to void, and its contents will be a virtual pointer to a memory buffer that will accept auto error sense information. The NWPA provides this field to support host adapter boards that do auto error sense under programmed I/O. When the CDM detects that auto error sense is active for a target device, it allocates an I/O contiguous buffer (using **NPA_Allocate_Memory**()) and assigns the buffer's NetWare logical address to this field. The structure of this buffer is defined by the NWPA's **ErrorSenseInfoStruct** plus any additional error sense bytes

the CDM chooses to append to this structure. Refer to the **ErrorSenseInfoStruct** reference information in Chapter 6 for more details about this concept. For adapters with auto error sense turned on, the HAM must copy auto error sense information into the buffer pointed at by this field.

pErrorSenseBufferPtr
This is a 4-byte field of type pointer to void. The NWPA calculates the physical (absolute) address of the buffer pointed at by **vDataBufferPtr** and places the address in this field. The NWPA provides the physical address to support adapters that use DMA or bus-mastering. Calculating this field value is not the concern of the CDM at HACB-build time. However, for safety, the CDM I/O routine building the HACB should initialize the field to zero. After the CDM I/O routine calls **CDI_Execute_HACB()**, the NWPA calculates the physical address and places it in this field before sending the HACB to the HAM.

Note: The NWPA guarantees this buffer to be physically contiguous

.
reserved1
This is a 6-LONG field reserved by NetWare

hamRserved This is a 64 -BYTE field reserved exclusively for private, HAM-specific use. This field may be used for anything necessary to complete the HACB request, such as linked list management of the HACB queue or custom command blocks such as disk structures, card structures, or control blocks.

Note:	As a reminder for the HAM, the HACB is not guaranteed to be below the 16 megabyte boundary, which may affect how this field can be used. Additionally, this field is uninitialized.

3.3.2.1 Union: Command Block Overlay Area

This section describes the different structures defined for the HACB's command overlay area (union). The NWPA defines the following types of command structures for this area: HOST-specific command structure (HACBType=0) SCSI-specific command structure (HACBType=1) IDE\ATA-specific command structure (HACBType=2) CDM Pass-Through command structure (HACBType=3) The CDM I/O routine that builds the HACB is responsible for selecting the appropriate structure, setting the structure's fields, and setting the HACBType field, all of which are based on the adapter type the CDM is designed to support. The HAM that receives the HACB can verify that the HACB is compatible with the

adapter type it is designed to support by reading the value in the HACBType field. The following subsections describe each structure and its fields.

3.3.2.1.1 Host Adapter Command Structure (HACBType=0) The Host Adapter Command Structure corresponds to HACBType=0 requests. This structure is used when adapter-specific commands are issued such as scanning for attached devices or getting adapter-specific information. Its field descriptions are as follows:

function
This is a 1-LONG field containing a function code, set by the CDM or the NWPA, that the HAM must map to a HAM function call. Table 3-3 maps the possible values for this field to their corresponding HAM functions. Full descriptions of these HAM functions can be found in Chapter 9, "HACB Type Zero Functions."

Table 3-3: Function Code Mapping of Type Zero HACB's to HAM

Field Value	HAM Function
0x0000	*HAM_Return_HAM_Info*
0x0001	*HAM_Scan_For_Devices*
0x0002	*HAM_Return_Device_Info*
0x0003	*HAM_Unfreeze_Queue*
0x0004	*HAM_Set_IDE_Drive_Config*
0x0005	*HAM_Queue_AEN_HACB*
0x0006	*HAM_Tag_Queue_Synch/Asynch*
0x0007 to 0x00FF	Reserved for future **HACBType=0** functions.

parameter0, parameter1, parameter2
These three fields are 1-LONG each (total of 3 LONGs) containing applicable type zero function parameters. See Chapter 8, "HACB Type Zero Functions" for a full description. These field values are set by the CDM or NWPA.

reserved2
This is a 12-BYTE field reserved by the NWPA

3.3.2.1.2 SCSI Adapter Command Structure (HACBType=1) The SCSI Adapter Command Structure corresponds to **HACBType=1** requests. This structure is used when a command is issued to a device attached to a SCSI adapter. Its field descriptions are as follows:

haCommandArea This is a 16-BYTE field containing the SCSI command that the HAM issues to the device. The CDM I/O routine that builds the HACB has the responsibility to set this field.

reserved3 This is a 11-BYTE field reserved by the NWPA.

haCommandLength This is a 1-BYTE field containing the device command length of the **haCommandArea** field. For SCSI, command lengths are either 6, 10, or 12 bytes. The CDM I/O routine that builds the HACB is responsible to set this field.

3.3.2.1.3 IDE\ATA Adapter Command Structure (HACBType=2) The IDE\ATA Adapter Command Structure corresponds to **HACBType=2** requests. This structure is used when a command is issued to a device attached to an IDE\ATA adapter. Its field descriptions are as follows:

numberSectorsRegister
This is a 1-BYTE field containing the value to be written to the IDE\ATA Sector Number Register. The CDM I/O routine that builds the HACB is responsible to set this field.

sectorRegister
This is a 1-BYTE field containing the value to be written to the IDE\ATA Sector Count Register. The CDM I/O routine that builds the HACB is responsible to set this field.

lowCylinderRegister
This is a 1-BYTE field containing the value to be written to the IDE\ATA Cylinder Low Register. The CDM I/O routine that builds the HACB is responsible to set this field.

highCylinderRegister
This is a 1-BYTE field containing the value to be written to the IDE\ATA Cylinder High Register. The CDM I/O routine that builds the HACB is responsible to set this field.

driveHeadRegister
This is a 1-BYTE field containing the value to be written to the IDE\ATA Drive/Head Register at command execution. The CDM I/O routine that builds the HACB is responsible to set this field.

commandRegister
This is a 1-BYTE field containing the value to be written to the IDE\ATA Command Register. The CDM I/O routine that builds the HACB is responsible to set this field.

reserved4
This is a 22-BYTE field reserved by the NWPA.

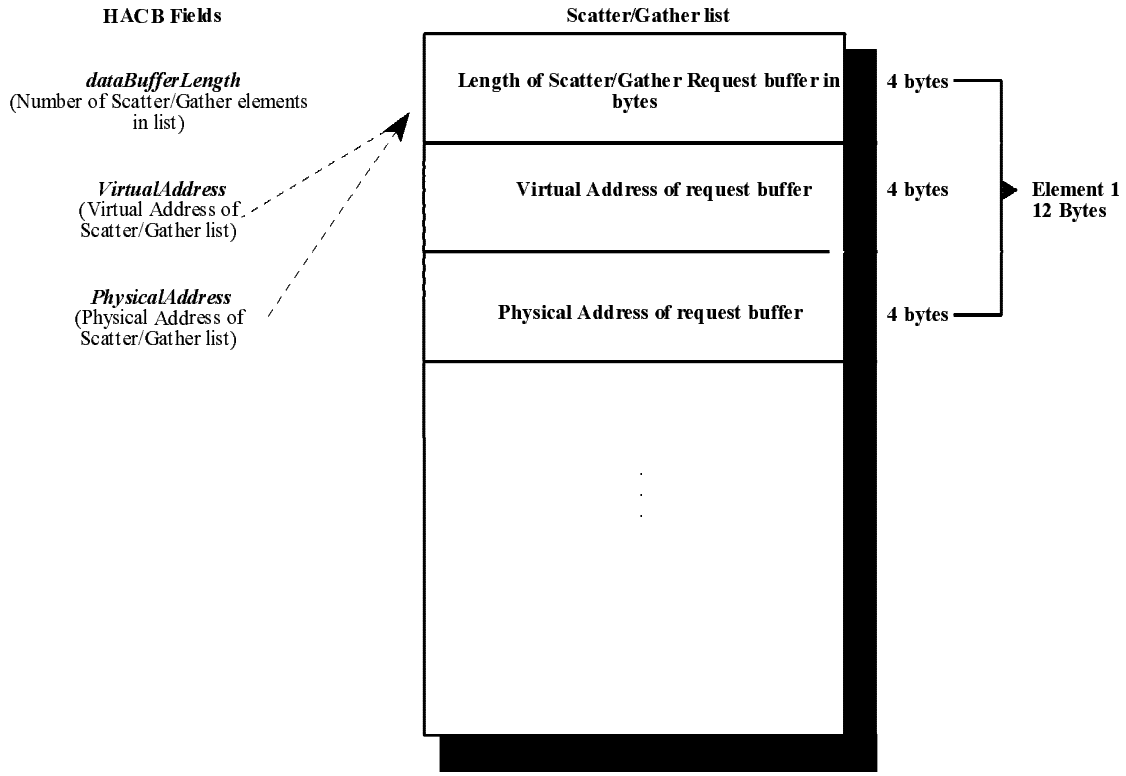
3.3.2.1.4 CDM Pass Through Command Structure (HACBType=3)
The CDM Pass-through Command Structure corresponds to **HACBType=3** requests. This structure is provided to support host adapters that support raw Media Manager functions. The HAM then has the responsibility to translate the HACB request information into the adapter-specific format and execute the request. The HAM also has the responsibility to translate the HACB completion status into a Media Manager-specific completion code and post this value to the HACB's **hacbCompletion** field instead of those listed in Appendix B. Essentially, the CDM I/O routine that builds the HACB copies the data in the CDM Message (**CDMMessageStruct**), field-for-field, into this pass-through structure and then passes it to the HAM by calling **CDI_Execute_HACB()**. Since the field data are the same, refer to "**CDMMessageStruct**" in Chapter 7 for a description of the fields.

3.4 Scatter/Gather List

This section specifies the format of the NWPA's scatter/gather request list. The term scatter/gather request is defined in the NWPA as a request that is contiguous on the device, but scattered in system memory. The NWPA will never issue a request in scatter/gather format unless the HAM supporting the device indicates that the host adapter to which the device is attached has scatter/gather capabilities. How the HAM makes this indication is discussed under the **AttributeFlags** field of the **DeviceInfoStruct** referenced in Chapter 7. The CDM determines whether a request is in scatter/gather format by masking the upper WORD of the CDM Message's (**CDMMessageStruct**) Function field. If ((UpperWORD & 0x0080) != 0) , then the request is in scatter/gather format, and the CDM must set the **Scatter_Gather_Flag** in the HACB's **Control_Info** field before sending it to the HAM. The HAM determines whether a HACB request is in scatter/gather format by checking the HACB's **Scatter_Gather_Flag** upon receipt of the HACB. The NWPA's scatter/gather list is actually a table that maps how the request is placed in system memory. In building a HACB for a scatter/gather request, the CDM simply places the information from the **CDMMessageStruct** into the appropriate fields of the HACB as described in the previous sections of this chapter. The HAM, however, must interpret the **dataBufferLength**, **vDataBufferPtr**, and **pDataBufferPtr** fields differently for a scatter/gather request. In the scatter/gather case, the

dataBufferLength field contains the number of elements in the scatter/gather request list. The **vDataBufferPtr** field contains the NetWare logical address of the scatter/gather list, and the **pDataBufferPtr** field contains the absolute, or physical, address of the scatter/gather list. Figure 3-1 illustrates this mapping:

Figure 3-1: NWPA Scatter/Gather List



The following are true for all scatter/gather requests: The minimum size of a scatter/gather request is 512 bytes. The scatter/gather buffer pointed to in memory is LONG aligned.

> **Important:** If hardware scatter/gather alignments on the host adapter are different than what is provided by the NWPA, the HAM must make the translation.