ADDENDUM

TO THE

NETWARE PERIPHERAL ARCHITECTURE (NWPA) FUNCTIONAL SPECIFICATION

AND DEVELOPER'S GUIDE

VERSION 2.1D (SEPTEMBER 1995)

Attached are changes to Version 2.1D of the NWPA Functional Specification. These changes are supported by Version 2.13 of NWPA.NLM These changes will be incorporated in the next release of the document.

1. Changes to **NPAOptionStruct** in Chapter 6.

These changes clarify the relationship between the parameters *parameter0*, *parameter1* and *parameter2* and the option type identified by the parameter *type*, and corrects information related to these parameters.

2. Addition of **NPAB_Get_Instance_Number**(), **NPAB_Get_Instance_Number_Mapping**(), and **NPAB_Get_Unique_Identifier_Parameters**().

These NBI related APIs are now supported by NWPA. The attachment provides their prototypes and definitions.

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 NWPA 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 NWPA 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 NWPA iteratively calls the HAM's *HAM_Check_Option()* routine for each option (provided that the options are of differing types) that was actually selected from the command line. *HAM_Check_Option()* can direct the NWPA to either accept the option by returning zero or reject the option by returning non-zero. If the option is accepted, the NWPA places it in a use list. The HAM then calls **NPA_Register_Options()** to direct the NWPA to physically register the options in its use list for the HAM.

The NWPA 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 add/parse/register 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[16];
} ;
```

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.

parameter0 This is a 4 byte field to contain the value associated with an option. See Table 6-1 for the relationship between this parameter and the option type selected.

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. See Table 6-1 for the relationship between this parameter and the option type selected. For the Product ID option, Figure 6-1 shows the various formats for product ID values (as applicable).

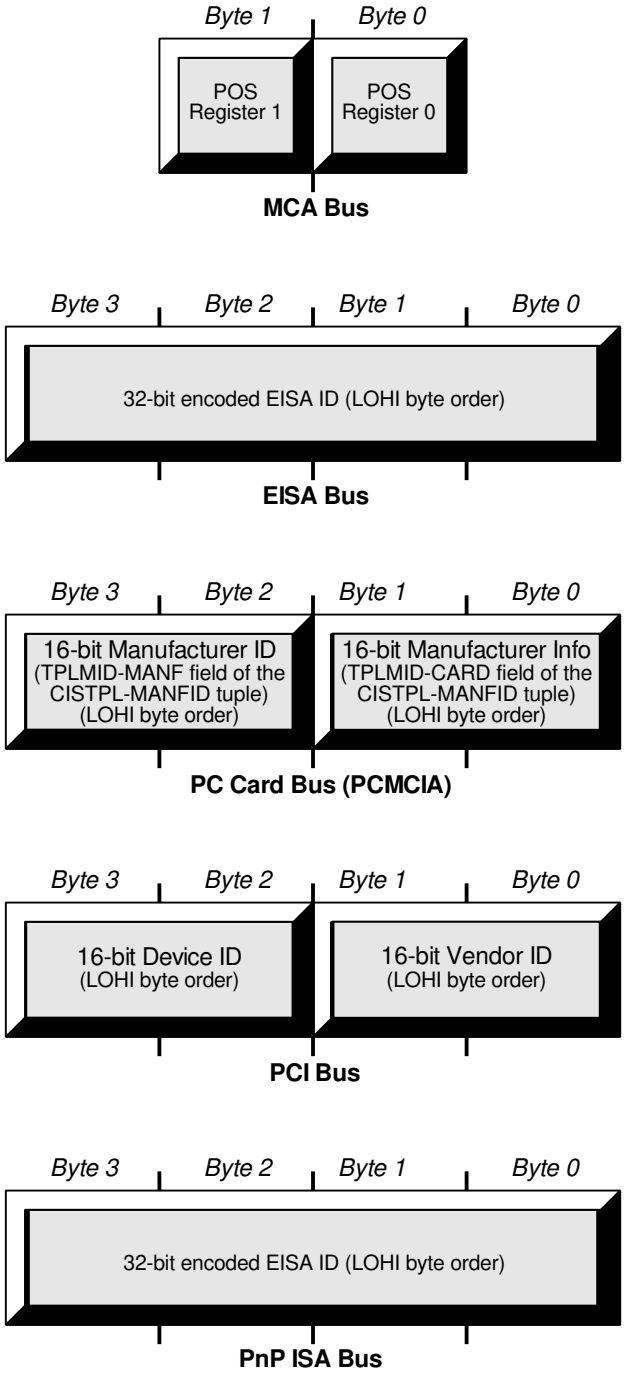
parameter2 This is a 4 byte field that can be either an input or an output parameter. See Table 6-1 for the relationship between this parameter and the option selected.

> **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. See Table 6-1 for the list of possible values for this field.

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 -- Tells NWPA to look for a "*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 -- Same as Value required except that instead of prompting the user for any value, a list of values will be displayed from which to choose. To use this flag value, developers must add a whole set of options of the same type with this flag set. From this group, NWPA will create its enumeration. Each value in the set is contained in *parameter0* of one of the options added in the set.
0x0010 Default value -- Contained in *parameter0*.
0x0020 Shareable option -- Such as shared interrupts.
All other bits in this field are reserved by NetWare.



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

Figure 6-1 Product ID Formats

string This is a 16-byte field that can be used to pass and\or receive information to/from the command line. If *flags* is set to Specific value required, this field will contain a length-preceded and NULL terminated string. *String* contains the ASCII code for the value (or values) 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, (*flags* is set to Value required) 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. When *flags* is set to Value required and the user was prompted to enter a value for this option, the entered ASCII string is placed in this field.

Table 6-1 Option Type vs. Parameter Definitions

Hex Value	Option	*parameter0*	*parameter1*	*parameter2*	
0x0000	HAM Defined option (such as NWDIAG)	Defined as needed by the option.	Defined as needed by the option.	Defined as needed by the option.	
0x0001	Interrupt Option	Contains the IRQ level.	0x01 -Put at end of ISR chain (Default is front of ISR chain.) 0x02- Adjust **RealModeInterrupt** mask. This enables real mode (DOS) Interrupts. 0x04- Reserved by NetWare.	**Input:** Contains the *busTag* that is returned by **NPAB_Get_Bus_Tag().** **Output:** Used by NetWare.	
0x0002	Port Option	Value of the port address being added.	Not used. Set to 0.	**Input:** Contains the *busTag* that is returned by **NPAB_Get_Bus_Tag().**	
0x0003	DMA Option	Value of the DMA channel being added.	Not used. Set to 0.	**Input:** Contains the *busTag* that is returned by **NPAB_Get_Bus_Tag().**	
0x0004	Memory decode option	Contains the shared memory absolute address used by the adapter for onboard me mory that is mapped into NetWare's logical address table.	Contains the number of paragraphs in the onboard memory.	**Input:** Contains the NBI defined *busTag* that is returned by **NPAB_Get_Bus_Tag().** **Output:**Contains the shared memory logical address of the mapped memory.	

Hex Value	Option	*parameter0*	*parameter1*	*parameter2*	
0x0005	Slot option	Value of the slot being added.	Not used. Set to 0.	**Input:** Contains the *busTag* that is returned by **NPAB_Get_Bus_Tag().**	
0x0006	Card option	Value of the card being added.	Not used. Set to 0.	Not used. Set to 0.	
0x0007	Reserved by NetWare	Reserved by NetWare.	Reserved by NetWare.	Reserved by NetWare.	
0x0008	Product ID option	**Input:** Contains the bus type returned from **NPAB_Get_Bus_Type().** **Output:** Contains the *busTag*.	**Input:** Contains a pointer to an array of bytes that contain a bus architecture-dependent parameter that uniquely identifies an adapter board/peripheral/system option. See Figure 6-1. **Output:** Contains the slot number.	**Input:** Contains the size of the array pointed to by *parameter1*. **Output:** Contains the *uniqueID*.	
0x0009 to 0x00FF	Reserved by NetWare	Reserved by NetWare.	Reserved by NetWare.	Reserved by NetWare.	
0x0100 to 0xFFFF	For vendor use as needed	For vendor use as needed.	For vendor use as needed	For vendor use as needed.	

Table 6-1 Option Type vs. Parameter Definitions (Continued)

NPAB_Get_Instance_Number

Purpose: Retrieves the instance number of the specified device or function on the specified bus.

Architecture Type: All

Thread Context: Non-Blocking

Requirements: None.

Syntax:

```
LONG NPAB_Get_Instance_Number(
    LONG   npaHandle,
    LONG   busTag,
    LONG   uniqueIdentifier,
    WORD   *instanceNumber);
```

Parameters:

Inputs:

npaHandle The HAM's handle for using the **NPA_** APIs, assigned during **NPA_Register_HAM_Module()**.

busTag Architecture dependent value returned by **NPAB_Get_Bus_Tag()**. It specifies the bus on which the operation is to be performed.

uniqueIdentifier Architecture dependent value returned by **NPAB_Get_Unique_Identifier()** or **NPAB_Search_Adapter()** that uniquely identifies a specific device or function.

Outputs:

instanceNumber Receives the instance number for the specified device or function. This value will be unique for all buses on the system.

Return Value: NBI_SUCCESSFUL - The requested operation was completed successfully.
NBI_PARAMETER_ERROR - One or more of the parameters passed was incorrect.

Description: There is a one-to-one correspondence betwen bus tag and unique identifier pairs and instance numbers. An instance number can be thought of as a logical slot number. If an adapter contains just one function, the instance number is equivalent to the adapter's physical slot number. Instance numbers are unique across all buses and devices on the system. They are generated or determined by NetWare and are consistent across system boots. The parameters *busTag* and *uniqueIdentifier* can also be obtained through adding and parsing the Product ID option (See **NPAOptionStruct** in Chapter 6.). The Product ID option will also return the *instanceNumber* (which is the same as "slot").

NPAB_Get_Instance_Number_Mapping

Purpose:

Retrieves the bus tag and unique identifier associated with the instance number specified.

Architecture Type: All

Thread Context: Non-Blocking

Requirements: None.

Syntax:

```
LONG NPAB_Get_Instance_Number_Mapping(
   LONG   npaHandle,
   WORD   instanceNumber,
   LONG   *busTag,
   LONG   *uniqueIdentifier);
```

Parameters:

Inputs:

npaHandle The HAM's handle for using the **NPA_** APIs, assigned during **NPA_Register_HAM_Module**().

instanceNumber The instance number of the device or function. The instance number is the slot parameter passed in the load-time command line.

Outputs:

busTag

Receives the bus tag for the selected instance number.

uniqueIdentifier

Receives the unique identifier for the selected instance number.

Return Value: NBI_SUCCESSFUL - The requested operation was completed successfully.
NBI_PARAMETER_ERROR - One or more of the parameters passed was incorrect.

Description: There is a one-to-one correspondence betwen bus tag and unique identifier pairs and instance numbers. An instance number can be thought of as a logical slot number. If an adapter contains just one function, the instance number is equivalent to the adapter's physical slot number. Instance numbers are unique across all buses and devices on the system. They are generated or determined by NetWare and are consistent across system boots.

NPAB_Get_Unique_Identifier_Parameters

Purpose: Returns bus-specific information about the device or function represented by the given unique identifier.

Architecture Type: All

Thread Context: Non-Blocking

Requirements: None.

Syntax:

```
LONG NPAB_Get_Unique_Identifier_Parameters(
    LONG   npaHandle,
    LONG   busTag,
    LONG   uniqueIdentifier,
    LONG   parameterCount,
    LONG   *parameters);
```

Parameters:

Inputs:

npaHandle The HAM's handle for using the **NPA_** APIs, assigned during **NPA_Register_HAM_Module()**.

busTag Architecture dependent value returned by **NPAB_Get_Bus_Tag()**. It specifies the bus on which the operation is to be performed.

uniqueIdentifier An architecture-dependent value returned by **NPAB_Search_Adapter()**, **NPAB_Get_Instance_Number_Mapping()**, **NPAB_Get_Unique_Identifier()**, or **NPAB_Scan_Card_Info()** that uniquely identifies a specific device or function.

parameterCount The number of elements in the parameter array to be filled in.

Outputs:

parameters An array of LONGs to be filled in with the bus architecture-dependent parameters represented by the specified unique identifier (See **NPAB_Get_Unique_Identifier()** for the format.)

Return Value: NBI_SUCCESSFUL- The requested operation was successful.
NBI_PARAMETER_ERROR- One or more of the parameters passed was incorrect.
NBI_UNSUPPORTED_OPERATION - The operation selected is not supported by NBI.

Description: **NPAB_Get_Unique_Identifier_Parameters()** is the inverse of **NPAB_Get_Unique_Identifier()**. This function returns bus-specific information for a given unique identifier. The parameters *busTag* and *uniqueIdentifier* can also be obtained through adding and parsing the Product ID option (See **NPAOptionStruct** in Chapter 6.). The Product ID option will also return the *instanceNumber* (which is the same as “slot”).