[MS-PNPR]:
Plug and Play Remote (PNPR) Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
12/18/2006	0.1		MCPP Milestone 2 Initial Availability
03/02/2007	1.1		MCPP Milestone 2
04/03/2007	1.2		Monthly release
05/11/2007	1.3		Monthly release
07/03/2007	1.3.1	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
07/20/2007	1.3.2	Editorial	Revised and edited the technical content.
08/10/2007	1.3.3	Editorial	Revised and edited the technical content.
09/28/2007	1.4	Minor	Updated the technical content.
10/23/2007	1.4.1	Editorial	Revised and edited the technical content.
11/30/2007	1.5	Minor	Updated, removed, and added some return codes.
01/25/2008	1.5.1	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This document specifies the Plug and Play Remote (PNPR) Protocol, a Microsoft proprietary **remote procedure call (RPC)** interface. The PNPR Protocol interface is used by the **client** for remote management of devices on the target system. The **server** does not maintain client state information. The protocol operation is stateless.

An RPC sequence is a client/server **session** that includes a security context phase and requests to call remote procedures. For **connection-oriented** RPC, the session also includes a binding phase.

The RPC client supplies the necessary security information; for connection-oriented RPC, the RPC client also supplies binding information such as interface name and server **endpoint**. The sequence of subsequent RPC calls in the session is implementation specific.

The request for information is provided through an RPC protocol interface. To receive this request, the server registers an endpoint using the **universal unique identifier (UUID)** 8D9F4E40-A03D-11CE-8F69-08003E30051B.

There are two versions of the Plug and Play Remote Protocol, Version 0.0 and Version 1.0. Version 0.0 contains a subset of the Version 1.0 functionality, in that all methods described and implemented in Version 0.0 are available in Version 1.0 with identical syntax. The increment in the version number results from the reassignment of **opnums** of the methods in Version 1.0, resulting in incompatibility of the two versions. This document specifies Version 1.0 of the PNPR Protocol.<1>

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Authenticated Users
Authentication Level
Authentication Service (AS)
Dynamic Endpoint
Endpoint
Globally Unique Identifier (GUID)
Interface Definition Language (IDL)
Microsoft Interface Definition Language (MIDL)
Network Data Representation (NDR)
Opnum
Remote Procedure Call (RPC)
RPC Protocol Sequence
RPC Transfer Syntax
RPC Transport
Security Provider
Server
Session
Universally Unique Identifier (UUID)
Well-Known Endpoint

The following terms are specific to this document:

Client: The computer on which the **RPC client** is executing.

Compatible ID: A vendor-defined identification string used to match a device to an INF file. **Compatible IDs** are used if a match for the **Hardware ID** is not found in an INF for a device.

Connection: An association established by a Server Message Block (SMB) **client** to a specific resource on an SMB **server** using a given **session**. See also **Session**.

Device Class GUID: A **globally unique identifier (GUID)** used to identify a category for grouping devices. The category referred to may either be a **Device Setup Class** or a **Device Interface Class**.

Device ID: A vendor-defined identification string describing a device.

Device Instance ID: A system-supplied device identification string that uniquely identifies a device in the system. A **device instance ID** is persistent across system boots.

Device Interface: Device functionality that a driver exposes to applications or other system components. Each **device interface** is a member of system-defined or vendor-defined **Device Interface Classes**. A driver can expose instances of zero, one, or more than one **Device Interface Classes** for a device. For example, a device can have a joystick and a keypad, and the device's driver stack can expose instances of three interface classes for the device: a joystick, a keypad, and a combined joystick/keypad.

Device Interface Class: A way of exporting device and driver functionality to other system components, including other drivers and user-mode applications. A driver can register a **Device Interface Class**, and then enable an instance of the class for each device object to which user-mode I/O requests might be sent. On the highest level, a **Device Interface Class** is a grouping of devices by functionality. A **GUID** uniquely identifies a **Device Interface Class**. Each **Device Interface Class** is associated with a **GUID**. The system defines **GUIDs** for common **Device Interface Classes**. Vendors can create additional **Device Interface Classes**.

Device Interface ID: A system-defined identification string that uniquely identifies a **device interface** in the system.

Device Setup Class: To facilitate device installation, devices that are set up and configured in the same way are grouped into a **Device Setup Class**. The **Device Setup Class** defines the class installer and the class co-installers that are involved in installing the device. A **GUID** uniquely identifies a **Device Setup Class**.

Docking Station: The base computer unit into which a user can insert a portable computer, expanding it to a desktop equivalent. A typical **docking station** provides drive bays, expansion slots, ports (desktop equivalents), and AC power.

Enumerator: A system component that discovers Plug and Play (PnP) devices based on a PnP hardware standard. For Windows 2000 and later, these tasks are carried out by PnP bus drivers in partnership with the PnP manager. A device is typically enumerated by its parent bus driver such as the PCI or PCMCIA bus driver. Some devices are enumerated by a bus filter driver such as the ACPI driver.

Enumerator ID: A vendor-defined identification string representing an **enumerator** on the system.

Hardware ID: A vendor-defined device identification string that is used to match a device to an INF file.Hardware Profile: A logical configuration of the machine that includes a set of instructions for what devices to start when the computer starts, or what settings to use for each device.

Hardware Resources: Low-level channels for transport of control signals to and from a device. For example, the IRQ (Interrupt Request Line) is a resource over which a peripheral device,

bus controller, other processor, or the kernel signals a request for **service** to the microprocessor.

Instance ID: A device identification string that distinguishes a device from other devices of the same type on a machine. An **instance ID** contains serial number information, if supported by the underlying bus, or some kind of location information.

Named Pipe: A named, one-way or duplex pipe for communication between a named **server** and one or more **named pipe clients**. For more information, see [PIPE].

Reference String: An optional, vendor-defined identification string used to differentiate between two interface instances of the same class for a single device.

Resource Descriptor: An entry in a resource requirements list, specifying a range of **hardware resources** that a device instance is capable of using; or an entry in a resource list, specifying a **hardware resource** assigned to a device instance.

Service: A string identifier representing a driver that can be associated with devices on the system. A driver **service** conforms to the device driver protocols. A driver **service** can be queried and configured through the **service** control manager interface, as specified in [MS-SCMR].

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, http://www.opengroup.org/public/pubs/catalog/c706.htm

[MS-DCOM] Microsoft Corporation, "Distributed Component Object Model (DCOM) Remote Protocol Specification", March 2007.

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", January 2007.

[MS-SCMR] Microsoft Corporation, "Service Control Manager Remote Protocol Specification", August 2007.

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol Specification", July 2007.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

1.2.2 Informative References

[PIPE] Microsoft Corporation, "Named Pipes", http://msdn2.microsoft.com/en-us/library/aa365590.aspx

1.3 Protocol Overview (Synopsis)

The Plug and Play Remote (PNPR) Protocol is designed for remotely querying and configuring devices on a remote computer. Using this protocol, a client can retrieve the set of devices on the server, discover and change settings for the devices, and query or configure the state of the devices.

This is an RPC-based protocol. The server does not maintain client state information. The protocol's operation is stateless

Using this protocol, it is possible to enumerate the list of devices and individually configure each device on remote systems on a network. It may be used by system administrators to query the health of devices on remote systems and maintain a list of currently active and functioning devices on the network. It also allows for remote configuration of devices for troubleshooting, enabling or disabling functionality, or for auditing purposes.

The server is implemented by the Plug and Play **service** and clients can connect to the server using a well-known **named pipe**. The client uses a well-known RPC named pipe **endpoint** to connect to the server.

Typically, communication between the client and the server begins with the client requesting a list of devices on the remote server. Each device is represented by a **device instance ID** string that identifies a specific device on the remote system. The client references these strings in subsequent calls to other methods implemented by the server to either query or modify the state of the device or to retrieve or set properties and attributes of the device. The client may also make configuration changes to the device, to enable or disable functionality of the device. The client may also enumerate the set of other abstract objects managed by the server, such as **device interfaces** or **device setup classes**, and similarly query or modify those objects.

Access and privilege checks to authenticate and authorize the client user are performed by the server before any of the methods available are executed.<2>

1.4 Relationship to Other Protocols

This protocol is dependent on RPC and Server Message Block (SMB) for its transport. This protocol uses RPC over named pipes, as specified in section 2.1. Named pipes use the Server Message Block (SMB) Protocol.

No protocols depend on this protocol.

1.5 Prerequisites/Preconditions

The Plug and Play Remote (PNPR) Protocol is an RPC interface and, as a result, has the prerequisites specified in [MS-RPCE] as common to RPC interfaces.

The server is started and fully initialized before the protocol starts.

It is assumed that a PNPR Protocol client has obtained the name of a remote machine that supports the Plug and Play Remote (PNPR) Protocol before this protocol is invoked. How a client does this is not addressed in this document.

1.6 Applicability Statement

The protocol described herein is applicable to environments that require discovery and configuration of devices on a remote computer.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- Supported Transports: This protocol uses RPC as a communication protocol. The only supported transport for this is Named Pipe over SMB (see section 2.1).
- Protocol Version: Two versions of this remote protocol have evolved over time, and the client and server are required to agree on the version number before communications proceed. Version 0.0 and Version 1.0 are incompatible and cannot be used interchangeably. The incompatibility arises from the rearrangement of opnums of the methods implemented in the two versions of the protocol. The version number can change, but this version of the protocol requires it to be a specific value (see section 2.1). This version of the protocol can be extended by adding RPC messages to the interface with opnums lying numerically beyond those defined in this specification. An RPC client determines if such methods are supported by attempting to invoke the method; if the method is not supported, the RPC runtime returns an "opnum out of range" error, as specified in [C706] and [MS-RPCE]. For RPC versioning and capacity negotiation in this situation, see [C706] and [MS-RPCE].
- Security and Authentication Methods: As specified in [MS-RPCE] and section 5.
- Capability Negotiation: This protocol's RPC interface may use either of two RPC well-known named pipe endpoints, as specified in section 2.1.

A client negotiates what well-known endpoint to use by first attempting to establish a connection to the server using the \\PIPE\plugplay well-known named pipe endpoint. If unsuccessful, the client attempts to establish a connection to the server using the \PIPE\ntsvcs well-known named pipe endpoint.

1.8 Vendor-Extensible Fields

This protocol uses string identifiers to uniquely identify devices and other abstract objects from all other such objects on the remote system. Vendors are free to choose their own values for these fields to identify devices provided that they conform to the guidelines for the string identifiers, as specified in section 2.2.8, and uniquely represent the objects on the server.

This protocol uses **globally unique identifiers (GUIDs)**, as specified in [MS-DTYP], to represent device class categories. Vendors are free to choose their own values for these fields to define new device class categories that may be assigned to devices.

This protocol uses string identifiers, GUIDs, and integer values as the data for many of the device registry property values specified in section 3.1.4.10. These values describe different attributes of a device or class. Vendors are free to choose their own values for the property data fields provided that the data conforms to the specification for the registry type values specified in section 3.1.4.10. The data MUST also be appropriate for the specific property as interpreted by the server and other clients.

This protocol uses **DEVPROPKEY (section 2.2.12.2)** structures to represent attributes of devices and other objects. Vendors are free to choose their own values for the fields in that structure to create new properties provided that they conform to the guidelines of the **DEVPROPKEY** (section 2.2.12.2) structure. The value of the data referenced by the **DEVPROPKEY** (section 2.2.12.2)

structures is also extensible by the vendor provided that the data conforms to the specification for the property type values (see section 3.1.4.57). The data MUST also be appropriate for the specific property as interpreted by the server and other clients.

1.9 Standards Assignments

Parameter	Value	Reference
Universally unique identifier (UUID)	8D9F4E40-A03D-11CE-8F69-08003E30051B	Section 6
Named pipe	\\PIPE\ntsvcs or \\PIPE\plugplay <3>	

2 Messages

The following sections specify how messages are transported and details of message syntax, including common structures, certificate requirements, and common error codes.

2.1 Transport

This protocol uses the following **RPC protocol sequence**: RPC over SMB, as specified in [MS-RPCE].

This protocol uses the following well-known endpoints. These endpoints are pipe names for RPC over SMB, as specified in [MS-RPCE] (for more information, see [PIPE]):

- \PIPE\ntsvcs
- \PIPE\plugplay

The \PIPE\ntsvcs well-known endpoint supports calls to interface methods with opnums lying numerically up to and including opnum 64, as supported by the server. This endpoint does not support any methods outside that range. Attempting to invoke the method, if the method is not supported, the RPC runtime returns an "opnum out of range" error, as specified in [C706] and [MS-RPCE]. This well-known endpoint SHOULD be implemented by all servers.

The \PIPE\plugplay well-known endpoint supports calls to all interface methods currently supported by the server. This well-known endpoint SHOULD be implemented by all servers. <4>

This protocol MUST use the UUID specified in section 1.9. The RPC version number is 1.0.

This protocol allows any user to establish a connection to the RPC server. The protocol uses the underlying RPC protocol to retrieve the identity of the caller that made the method call, as specified in [MS-RPCE] section 3.3.3.4.3. The server SHOULD use this identity to perform method-specific access checks, as specified in section 3.1.4.

2.2 Common Data Types

This protocol MUST indicate to the RPC runtime that it is to support the **Network Data Representation (NDR)** 20 (that is, NDR20) transfer syntax only, as specified in [C706] part 4.

This protocol MUST enable the ms_union extension, as specified in [MS-RPCE] section 2.2.4.

In addition to RPC base types and definitions specified in [C706] and [MS-DTYP], additional data types are defined below.

2.2.1 PNP_PROP_SIZE

The PNP_PROP_SIZE datatype is used to specify the size, in bytes, of a PropertyBuffer.

This type is declared as follows:

```
typedef [range(0,PNP_MAX_PROP_SIZE)]
  unsigned long PNP_PROP_SIZE;
```

2.2.2 PPNP_PROP_SIZE

The PPNP_PROP_SIZE datatype is a pointer to a **PNP_PROP_SIZE** datatype.

This type is declared as follows:

```
typedef PNP_PROP_SIZE* PPNP_PROP_SIZE;
```

2.2.3 RESOURCEID

The RESOURCEID datatype is used to specify a RESOURCEID data type.

This type is declared as follows:

```
typedef unsigned long RESOURCEID;
```

2.2.4 PNP_RPC_STRING_LEN

The PNP_RPC_STRING_LEN datatype is used to specify the size, in characters, of the string buffer.

This type is declared as follows:

```
typedef unsigned long PNP_RPC_STRING_LEN;
```

2.2.5 PNP_RPC_BUFFER_SIZE

The PNP_RPC_BUFFER_SIZE datatype is used to specify the size, in characters, of a buffer.

This type is declared as follows:

```
typedef unsigned long PNP_RPC_BUFFER_SIZE;
```

2.2.6 PNP_PROP_COUNT

The PNP_PROP_COUNT datatype is used to specify the number of **DEVPROPKEY** elements.

This type is declared as follows:

```
typedef unsigned long PNP_PROP_COUNT;
```

2.2.7 DEVPROPTYPE

The DEVPROPTYPE is used to specify property types,each of which requires a different format for a corresponding property buffer.

This type is declared as follows:

```
typedef unsigned long DEVPROPTYPE;
```

2.2.8 Abstract Data Elements

The PNPR Protocol interface sets the following limits on the format of abstract data elements.

2.2.8.1 Device ID

A **device ID** string MUST have one of the following generic formats:

```
<enumerator>\<enumerator-specific-device-ID>
*<enumerator-specific-ID>
<device-class-specific-ID>
```

A device ID string MUST NOT contain any of the following invalid characters:

```
c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
```

The number of characters of a device ID, excluding a NULL terminator, MUST be less than 200. This constraint applies to the sum of the lengths of all the fields and any "\" field separators in a device ID. In addition, when an **instance ID** is concatenated to a device ID to create a device instance ID, the lengths of the device ID and the instance ID are further constrained by the maximum possible length of a device instance ID.

2.2.8.2 Compatible ID

A **compatible ID** string MUST have the same format as what is specified for a device ID.

The number of characters in a list of compatible ID strings (including a NULL terminator after each compatible ID and a final NULL terminator) MUST be less than 1,024. The number of compatible ID strings in a list of compatible IDs MUST be less than 64.

2.2.8.3 Device Class GUID String ID

A **Device Class GUID** string ID MUST be specified in the format defined for a GUID string, as specified in [MS-GLOS].

A GUID string is the string representation of a 128-bit GUID, using the form {XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX} where X denotes a hexadecimal digit.

2.2.8.4 Device Instance ID

A device instance ID string MUST have the following format:

```
<enumerator-ID>\<enumerator-specific-device-ID>
\<instance-specific-ID>
```

The number of characters of a device instance ID, including a NULL-terminator, MUST be less than 200 characters. This constraint applies to the sum of the lengths of all the fields and the "\" field separator between the device ID and the instance-specific ID fields.

2.2.8.5 Device Interface ID

A **device interface ID** string MUST have one of the following formats:

```
\\?\<device-specific-interface-ID>
\\?\<device-specific-interface-ID>\<reference-ID>
```

The device-specific interface ID is generated by the system. The **reference string** component is optional. If no reference string is present, the "\" field separator character that would precede it MUST not be present.

A device interface ID string MUST NOT contain any of the following invalid characters:

```
c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
```

The number of characters of a device interface ID, excluding a NULL terminator, MUST be less than 32,767.

2.2.8.6 Enumerator ID

An **Enumerator ID** string MUST not contain any "\" field separators or any of the following invalid characters:

```
c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
```

The number of characters of an enumerator ID, excluding a NULL terminator, MUST be less than 200. In addition, when an **enumerator** device ID string is concatenated to an enumerator ID to create a device ID, the lengths of the enumerator ID and enumerator device ID are further constrained by the maximum possible length of a device ID.

2.2.8.7 Hardware ID

A **hardware ID** string MUST have the same format as what is specified for a device ID.

The number of characters in a list of hardware ID strings (including a NULL terminator after each hardware ID and a final NULL terminator) MUST be less than 1,024. The number of hardware ID strings in a list of hardware IDs MUST be less than 64.

2.2.8.8 Instance ID

An instance ID string MUST NOT contain any "\" characters; otherwise, the generic format of the string is bus specific.

The number of characters of an instance ID, excluding a NULL-terminator, MUST be less than 200. In addition, when an instance ID is concatenated to a device ID to create a device instance ID, the lengths of the device ID and the instance ID are further constrained by the maximum possible length of a device instance ID.

2.2.8.9 Reference String

A reference string ID string MUST NOT contain any "\" field separators or any of the following invalid characters:

```
c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
```

The number of characters of a reference string ID, excluding a NULL terminator, MUST be less than 260. In addition, when a reference string ID string is concatenated to a device-specific interface ID to create a device interface ID, the length of the reference string ID is further constrained by the maximum possible length of a device interface ID.

2.2.9 Constants

The Plug and Play Remote (PNPR) Protocol sets the following limits on the interface method parameters. <5>

Constant/value	Description
PNP_MAX_STRING_LEN 32767	The maximum length of a NULL-terminated UNICODE string in characters, including the ending NULL character.
PNP_MAX_DEVICE_ID_LEN 200	The maximum length of a NULL-terminated device instance ID string in characters, including the NULL character.
PNP_MAX_GUID_STRING_LEN 39	The maximum length of a NULL-terminated GUID string in characters, including the NULL character.
PNP_MAX_DEVINTERFACE_LEN 32767	The maximum length of a NULL-terminated device interface string in characters, including the NULL character.
PNP_MAX_CULTURE_NAME_LEN 85	The maximum length of a NULL-terminated culture name string in characters, including the NULL character.
PNP_MAX_CM_PATH 360	The maximum length of the NULL-terminated registry path string in characters, including the NULL character that references the location that stores device configuration information.
PNP_MAX_PROP_SIZE 65534	The maximum size of a property value in bytes.
PNP_MAX_PROP_COUNT	The maximum number of properties that may be returned by the pnp

Constant/value	Description
32767	interface.
PNP_MAX_BUFFER_SIZE 0xF42400	The maximum size of the buffer, in bytes, that can be allocated by the PNPR Protocol interface.

2.2.10 Data Types

```
typedef [range(0,PNP_MAX_PROP_SIZE)]
  unsigned long PNP_PROP_SIZE;

typedef [range(0,PNP_MAX_PROP_COUNT)]
  unsigned long PNP_PROP_COUNT, *PPNP_PROP_COUNT;

typedef [range(0,PNP_MAX_STRING_LEN)]
  unsigned long PNP_RPC_STRING_LEN, *PPNP_RPC_STRING_LEN;

typedef [range(0,PNP_MAX_BUFFER_SIZE)]
  unsigned long PNP_RPC_BUFFER_SIZE, *PPNP_RPC_BUFFER_SIZE;
```

PNP_PROP_SIZE

The size of a property.

PNP_PROP_COUNT

The number of properties returned by the pnp interface

PNP_RPC_STRING_LEN

Length of a NULL-terminated UNICODE string in characters, including the ending NULL character.

PNP_RPC_BUFFER_SIZE

Size of the buffer, in bytes, that can be allocated by the PNPR Protocol interface.

2.2.11 Enumerations

2.2.11.1 PPNP_VETO_TYPE

The **PPNP_VETO_TYPE** enumeration is used to indicate the type of component on the server responsible for vetoing a configuration change request such as the removal of a device.

```
typedef  enum
{
  PNP_VetoTypeUnknown = 0,
  PNP_VetoLegacyDevice = 1,
  PNP_VetoPendingClose = 2,
  PNP_VetoWindowsApp = 3,
  PNP_VetoWindowsService = 4,
  PNP_VetoOutstandingOpen = 5,
```

```
  PNP_VetoDevice = 6,
  PNP_VetoDriver = 7,
  PNP_VetoIllegalDeviceRequest = 8,
  PNP_VetoInsufficientPower = 9,
  PNP_VetoNonDisableable = 10,
  PNP_VetoLegacyDriver = 11,
  PNP_VetoInsufficientRights = 12
}  *PPNP_VETO_TYPE;
```

PNP_VetoTypeUnknown: Used when none of the others apply.

PNP_VetoLegacyDevice: Vetoed by a legacy (non-PnP) device or a device with a legacy driver.

PNP_VetoPendingClose: Not used.

PNP_VetoWindowsApp: Vetoed by an application.

PNP_VetoWindowsService: Named service vetoed the operation.

PNP_VetoOutstandingOpen: Vetoed due to open handles on the device.

PNP_VetoDevice: Vetoed by the device driver.

PNP_VetoDriver: Vetoed by the driver.

PNP_VetoIllegalDeviceRequest: Device is not removable; thus, the request is illegal.

PNP_VetoInsufficientPower: There is insufficient power to complete the operation.

PNP_VetoNonDisableable: Device cannot be disabled. It may be needed to boot the system, for example.

PNP_VetoLegacyDriver: Vetoed due to a legacy driver; that is, a driver that called one of the legacy resource allocation APIs.

PNP_VetoInsufficientRights: Insufficient privilege held by caller to complete the operation.

2.2.12 Structures

2.2.12.1 Resource Structures

The **Resource Descriptor** structures are used to specify the resources required or resources that have been assigned to a device instance. Multiple resource descriptors are grouped into logical configurations. A device can have multiple logical configurations representing different types of configurations, or multiple possible configurations for the device.

The Resource Descriptor structures use 1-byte packing, and are transmitted by the protocol within an unstructured byte array buffer parameter.

2.2.12.1.1 BUSNUMBER Structures

The BUSNUMBER structures correspond to the ResType_BusNumber resource type specified in section 3.1.4.44.

2.2.12.1.1.1 BUSNUMBER_DES

The **BUSNUMBER_DES** structure is used for specifying either a resource list or a resource requirements list that describes bus number usage for a device instance.

```
typedef struct _BUSNUMBER_DES {
  DWORD BUSD_Count;
  DWORD BUSD_Type;
  DWORD BUSD_Flags;
  unsigned long BUSD_Alloc_Base;
  unsigned long BUSD_Alloc_End;
} BUSNUMBER_DES,
 *PBUSNUMBER_DES;
```

BUSD_Count: The number of **BUSNUMBER_RANGE (section 2.2.12.1.1.2)** structures in **BUSNUMBER_RESOURCE**.

BUSD_Type: MUST be set to the value of the size of the **BUSNUMBER_RANGE** (section 2.2.12.1.1.2) structure.

BUSD_Flags: Flags describing the range (currently unused).

BUSD_Alloc_Base: The lowest number of a range of contiguous bus numbers allocated to the device.

BUSD_Alloc_End: The highest number of a range of contiguous bus numbers allocated to the device.

2.2.12.1.1.2 BUSNUMBER_RANGE

The **BUSNUMBER_RANGE** structure specifies a resource requirements list that describes bus number usage for a device instance.

```
typedef struct _BUSNUMBER_RANGE {
  unsigned long BUSR_Min;
  unsigned long BUSR_Max;
  unsigned long BUSR_nBusNumbers;
  unsigned long BUSR_Flags;
} BUSNUMBER_RANGE,
 *PBUSNUMBER_RANGE;
```

BUSR_Min: The lowest number of a range of contiguous bus numbers that can be allocated to the device.

BUSR_Max: The highest number of a range of contiguous bus numbers that can be allocated to the device.

BUSR_nBusNumbers: The number of contiguous bus numbers required by the device.

BUSR_Flags: Flags describing the range (currently unused).

2.2.12.1.1.3 BUSNUMBER_RESOURCE

The **BUSNUMBER_RESOURCE** structure specifies either a resource list or a resource requirements list that describes bus number usage for a device instance.

```
typedef struct _BUSNUMBER_RESOURCE {
  BUSNUMBER_DES BusNumber_Header;
  BUSNUMBER_RANGE BusNumber_Data[1];
} BUSNUMBER_RESOURCE,
 *PBUSNUMBER_RESOURCE;
```

BusNumber_Header: Information on the bus number range list.

BusNumber_Data: List of bus number ranges.

2.2.12.1.2 CS Structures

The CS structures correspond to the ResType_ClassSpecific resource type, as specified in section 3.1.4.44.

2.2.12.1.2.1 CS_DES

The **CS_DES** structure is used for specifying a resource list that describes device class-specific resource usage for a device instance.

```
typedef struct _CS_DES {
  DWORD CSD_SignatureLength;
  DWORD CSD_LegacyDataOffset;
  DWORD CSD_LegacyDataSize;
  DWORD CSD_Flags;
  GUID CSD_ClassGuid;
  byte CSD_Signature[1];
} CS_DES,
 *PCS_DES;
```

CSD_SignatureLength: The number of elements in the byte array specified by CSD_Signature.

CSD_LegacyDataOffset: Offset, in bytes, from the beginning of the CSD_Signature array to the beginning of a block of data. For example, if the data block follows the signature array, and if the signature array length is 16 bytes, the value for CSD_LegacyDataOffset should be 16.

CSD_LegacyDataSize: Length, in bytes, of the data block whose offset is specified by CSD_LegacyDataOffset.

CSD_Flags: Not used.

CSD_ClassGuid: A globally unique identifier (GUID) identifying a Device Setup Class. If both CSD_SignatureLength and CSD_LegacyDataSize are zero, the GUID is null.

CSD_Signature: A byte array containing a class-specific signature. The number of elements in the byte array is specified by CSD_SignatureLength.

2.2.12.1.2.2 CS_RESOURCE

The **CS_RESOURCE** structure is used for specifying a resource list that describes device class-specific resource usage for a device instance.

```
typedef struct _CS_RESOURCE {
  CS_DES CS_Header;
} CS_RESOURCE,
 *PCS_RESOURCE;
```

CS_Header: A CS_DES (section 2.2.12.1.2.1) structure.

2.2.12.1.3 DevicePrivate Structures

The DevicePrivate structures correspond to the ResType_DevicePrivate resource type, as specified in section 3.1.4.44.

2.2.12.1.3.1 DEVPRIVATE_DES

The **DEVPRIVATE_DES** structure is used for specifying either a resource list or a resource requirements list that describes private device-specific resource usage for a device instance.

```
typedef struct _DEVPRIVATE_DES {
  DWORD PD_Count;
  DWORD PD_Type;
  DWORD PD_Data1;
  DWORD PD_Data2;
  DWORD PD_Data3;
  DWORD PD_Flags;
} DEVPRIVATE_DES,
 *PDEVPRIVATE_DES;
```

PD_Count: The number of DEVPRIVATE_RANGE (section 2.2.12.1.3.2) structures in DEVPRIVATE_RESOURCE.

PD_Type: MUST be set to the value of the size of the **DEVPRIVATE_RANGE** (section 2.2.12.1.3.2) structure.

PD_Data1: Specifies device-specific resource data.

PD_Data2: Specifies device-specific resource data.

PD_Data3: Specifies device-specific resource data.

PD_Flags: Flags describing the range (currently unused).

2.2.12.1.3.2 DEVPRIVATE_RANGE

The **DEVPRIVATE_RANGE** structure specifies a resource requirements list that describes private device-specific resource usage for a device instance.

```
typedef struct _DEVPRIVATE_RANGE {
```

```
  DWORD PR_Data1;
  DWORD PR_Data2;
  DWORD PR_Data3;
} DEVPRIVATE_RANGE,
 *PDEVPRIVATE_RANGE;
```

PR_Data1: Specifies device-specific resource range data.

PR_Data2: Specifies device-specific resource range data.

PR_Data3: Specifies device-specific resource range data.

2.2.12.1.3.3 DEVPRIVATE_RESOURCE

The **DEVPRIVATE_RESOURCE** structure specifies a resource requirements list that describes private device-specific resource usage for a device instance.

```
typedef struct _DEVPRIVATE_RESOURCE {
  DEVPRIVATE_DES PRV_Header;
  DEVPRIVATE_RANGE PRV_Data[1];
} DEVPRIVATE_RESOURCE,
 *PDEVPRIVATE_RESOURCE;
```

PRV_Header: A **DEVPRIVATE_DES (section 2.2.12.1.3.1)** structure.

PRV_Data: A **DEVPRIVATE_RANGE (section 2.2.12.1.3.2)** array.

2.2.12.1.4 DMA Structures

The DMA structures correspond to the ResType_DMA resource type, as specified in section 3.1.4.44.

2.2.12.1.4.1 DMA_DES

The **DMA_DES** structure is used for specifying either a resource list or a resource requirements list that describes direct memory access (DMA) channel usage for a device instance.

```
typedef struct _DMA_DES {
  DWORD DD_Count;
  DWORD DD_Type;
  DWORD DD_Flags;
  unsigned long DD_Alloc_Chan;
} DMA_DES,
 *PDMA_DES;
```

DD_Count: For a resource list: Zero. For a resource requirements list: The number of elements in the **DMA_RANGE (section 2.2.12.1.4.2)** array that is included in the **DMA_RESOURCE (section 2.2.12.1.4.3)** structure.

DD_Type: MUST be set to the value of the size of the **DMA_RANGE** (section 2.2.12.1.4.2) structure.

DD_Flags: A bit flag from each of the flag sets defined in the following table.

Channel Width Flags

Value	Meaning
fDD_BYTE 0x00000000	8-bit DMA channel.
fDD_WORD 0x00000001	16-bit DMA channel.
fDD_DWORD 0x00000002	32-bit DMA channel.
fDD_BYTE_AND_WORD 0x00000003	8-bit and 16-bit DMA channel.

The bitmask for the bits within DD_Flags that specify the channel width value is 0x00000003 (mDD_Width).

Bus Mastering Flags

Value	Meaning
fDD_NoBusMaster 0x00000000	No bus mastering.
fDD_BusMaster 0x00000004	Bus mastering.

The bitmask for the bits within DD_Flags that specify the bus mastering value is 0x00000004 (mDD_BusMaster).

DMA Type Flags

Value	Meaning
fDD_TypeStandard 0x00000000	Standard DMA.
fDD_TypeA 0x00000008	Type A DMA.
fDD_TypeB 0x00000010	Type B DMA.
fDD_TypeF 0x00000018	Type F DMA.

The bitmask for the bits within DD_Flags that specify the DMA type value is 0x00000018 (mDD_Type).

DD_Alloc_Chan: For a resource list: The DMA channel allocated to the device. For a resource requirements list: Not used.

2.2.12.1.4.2 DMA_RANGE

The **DMA_RANGE** structure specifies a resource requirements list that describes DMA channel usage for a device instance.

```
typedef struct _DMA_RANGE {
  unsigned long DR_Min;
  unsigned long DR_Max;
  unsigned long DR_Flags;
} DMA_RANGE,
 *PDMA_RANGE;
```

DR_Min: The lowest-numbered DMA channel that can be allocated to the device.

DR_Max: The highest-numbered DMA channel that can be allocated to the device.

DR_Flags: A bit flag from each of the flag sets defined in the table included with the description of the DR_Flags member of the **DMA_DES (section 2.2.12.1.4.1)** structure.

2.2.12.1.4.3 DMA_RESOURCE

The **DMA_RESOURCE** structure is used for specifying either a resource list or a resource requirements list that describes DMA channel usage for a device instance.

```
typedef struct _DMA_RESOURCE {
  DMA_DES DMA_Header;
  DMA_RANGE DMA_Data[1];
} DMA_RESOURCE,
 *PDMA_RESOURCE;
```

DMA_Header: A **DMA_DES (section 2.2.12.1.4.1)** structure.

DMA_Data: For a resource list: Zero. For a resource requirements list: A **DMA_RANGE (section 2.2.12.1.4.2)** array.

2.2.12.1.5 IO Structures

The IO structures correspond to the ResType_IO resource type, as specified in section 3.1.4.44.

2.2.12.1.5.1 IO_DES

The **IO_DES** structure is used for specifying either a resource list or a resource requirements list that describes I/O port usage for a device instance.

```
typedef struct _IO_DES {
  DWORD IOD_Count;
  DWORD IOD_Type;
  unsigned __int64 IOD_Alloc_Base;
  unsigned __int64 IOD_Alloc_End ;
  DWORD IOD_DesFlags;
} IO_DES,
 *PIO_DES;
```

IOD_Count: For a resource list: Zero. For a resource requirements list: The number of elements in the **IO_RANGE (section 2.2.12.1.5.2)** array that is included in the **IO_RESOURCE**.

IOD_Type: MUST be set to the value of the size of the **IO_RANGE** (section 2.2.12.1.5.2) structure.

IOD_Alloc_Base: For a resource list: The lowest-numbered of a range of contiguous I/O port addresses allocated to the device. For a resource requirements list: Zero.

IOD_Alloc_End : For a resource list: The highest-numbered of a range of contiguous I/O port addresses allocated to the device. For a resource requirements list: Zero.

IOD_DesFlags: A bit flag from each of the flag sets defined in the following table.

Port Type Flags

Value	Meaning
fIOD_IO 0x00000001	The device is accessed in I/O address space.
fIOD_Memory 0x00000000	The device is accessed in memory address space.

The bitmask for the bits within IOD_DesFlags that specify the port type value is 0x00000001 (fIOD_PortType).

Decode Flags

Value	Meaning
fIOD_10_BIT_DECODE 0x00000004	The device decodes 10 bits of the port address.
fIOD_12_BIT_DECODE 0x00000008	The device decodes 12 bits of the port address.
fIOD_16_BIT_DECODE 0x00000010	The device decodes 16 bits of the port address.
fIOD_POSITIVE_DECODE 0x00000020	The device uses positive decode instead of subtractive decode.

The bitmask for the bits within IOD_DesFlags that specify the decode value is 0x000000FC (fIOD_DECODE).

2.2.12.1.5.2 IO_RANGE

The **IO_RANGE** structure specifies a resource requirements list that describes I/O port usage for a device instance.

```
typedef struct _IO_RANGE {
  unsigned __int64 IOR_Align;
```

```
  DWORD IOR_nPorts;
  unsigned __int64 IOR_Min;
  unsigned __int64 IOR_Max;
  DWORD IOR_RangeFlags;
  unsigned __int64 IOR_Alias;
} IO_RANGE,
 *PIO_RANGE;
```

IOR_Align: Mask used to specify the port address boundary on which the first allocated I/O port address must be aligned.

IOR_nPorts: The number of I/O port addresses required by the device.

IOR_Min: The lowest numbered of a range of contiguous I/O port addresses that can be allocated to the device.

IOR_Max: The highest numbered of a range of contiguous I/O port addresses that can be allocated to the device.

IOR_RangeFlags: A bit flag from each of the flag sets defined in the table included with the description of the IOD_DesFlags member of the **IO_DES (section 2.2.12.1.5.1)** structure.

IOR_Alias: One of the bit flags defined in the following table.

Value	Meaning
IO_ALIAS_10_BIT_DECODE 0x00000004	The device decodes 10 bits of the port address.
IO_ALIAS_12_BIT_DECODE 0x00000010	The device decodes 12 bits of the port address.
IO_ALIAS_16_BIT_DECODE 0x00000000	The device decodes 16 bits of the port address.
IO_ALIAS_POSITIVE_DECODE 0x000000FF	The device uses positive decode instead of subtractive decode.

The flags specified for IOR_Alias have the same interpretation as the address decoding flags specified for IOD_DesFlags. (However, the two sets of flags are not equivalent in assigned values and cannot be used interchangeably.) A resource requirements list can be specified using either set of flags, but using decode flags in IOD_DesFlags is recommended. If address decoding flags are specified using both IOD_DesFlags and IOR_Alias, the contents of the latter overrides the former.

2.2.12.1.5.3 IO_RESOURCE

The **IO_RESOURCE** structure is used for specifying either a resource list or a resource requirements list that describes I/O port usage for a device instance.

```
typedef struct _IO_RESOURCE {
  IO_DES IO_Header;
  IO_RANGE IO_Data[1];
} IO_RESOURCE,
```

```
*PIO_RESOURCE;
```

IO_Header: An **IO_DES (section 2.2.12.1.5.1)** structure.

IO_Data: For a resource list: Zero. For a resource requirements list: An **IO_RANGE (section 2.2.12.1.5.2)** array.

2.2.12.1.6 IRQ Structures

The IRQ structures correspond to the ResType_IRQ resource type, as specified in section 3.1.4.44.

2.2.12.1.6.1 IRQ_DES

The **IRQ_DES** structure is used for specifying either a resource list or a resource requirements list that describes IRQ line usage for a device instance.

```
typedef struct _IRQ_DES {
  DWORD IRQD_Count;
  DWORD IRQD_Type;
  DWORD IRQD_Flags;
  unsigned long IRQD_Alloc_Num;
  unsigned long IRQD_Affinity;
} IRQ_DES,
 *PIRQ_DES;
```

IRQD_Count: For a resource list: Zero. For a resource requirements list: The number of elements in the **IRQ_RANGE (section 2.2.12.1.6.2)** array that is included in the **IRQ_RESOURCE (section 2.2.12.1.6.3)** structure.

IRQD_Type: MUST be set to the value of the size of the **IRQ_RANGE** (section 2.2.12.1.6.2) structure.

IRQD_Flags: A bit flag from each of the flag sets defined in the following table.

Sharing Flags

Value	Meaning
fIRQD_Exclusive 0x00000000	The IRQ line cannot be shared.
fIRQD_Share 0x00000001	The IRQ line can be shared.

The bitmask for the bits within IRQD_Flags that specify the sharing value is 0x00000001 (mIRQD_Share).

Triggering Flags

Value	Meaning
fIRQD_Level 0x00000000	The IRQ line is level triggered.
fIRQD_Edge 0x00000002	The IRQ line is edge triggered.

The bitmask for the bits within IRQD_Flags that specify the triggering value is 0x00000002 (mIRQD_Edge_Level).

IRQD_Alloc_Num: For a resource list: The number of the IRQ line that is allocated to the device. For a resource requirements list: Not used.

IRQD_Affinity: For a resource list: A bitmask representing the processor affinity of the IRQ line allocated to the device. Bit zero represents the first processor, bit two the second, and so on. Set this value to -1 to represent all processors. For a resource requirements list: Not used.

2.2.12.1.6.2 IRQ_RANGE

The **IRQ_RANGE** structure specifies a resource requirements list that describes IRQ line usage for a device instance.

```
typedef struct _IRQ_RANGE {
  unsigned long IRQR_Min;
  unsigned long IRQR_Max;
  unsigned long IRQR_Flags;
} IRQ_RANGE,
 *PIRQ_RANGE;
```

IRQR_Min: The lowest numbered of a range of contiguous IRQ lines that can be allocated to the device.

IRQR_Max: The highest numbered of a range of contiguous IRQ lines that can be allocated to the device.

IRQR_Flags: A bit flag from each of the flag sets defined in the table included with the description of the IRQD_Flags member of the **IRQ_DES (section 2.2.12.1.6.1)** structure.

2.2.12.1.6.3 IRQ_RESOURCE

The **IRQ_RESOURCE** structure is used for specifying either a resource list or a resource requirements list that describes IRQ line usage for a device instance.

```
typedef struct _IRQ_RESOURCE {
  IRQ_DES IO_Header;
  IRQ_RANGE IO_Data[1];
} IRQ_RESOURCE,
 *PIRQ_RESOURCE;
```

IO_Header: An **IRQ_DES (section 2.2.12.1.6.1)** structure.

IO_Data: For a resource list: Zero. For a resource requirements list: An **IRQ_RANGE (section 2.2.12.1.6.2)** array.

2.2.12.1.7 Mem Structures

The Mem structures correspond to the ResType_Mem resource type, as specified in section 3.1.4.44.

2.2.12.1.7.1 MEM_DES

The **MEM_DES** structure is used for specifying either a resource list or a resource requirements list that describes memory usage for a device instance.

```
typedef struct _MEM_DES {
  DWORD MD_Count;
  DWORD MD_Type;
  unsigned __int64 MD_Alloc_Base;
  unsigned __int64 MD_Alloc_End;
  DWORD MD_Flags;
  DWORD Reserved;
} MEM_DES,
 *PMEM_DES;
```

MD_Count: For a resource list: Zero. For a resource requirements list: The number of elements in the **MEM_RANGE (section 2.2.12.1.7.2)** array that is included in the **MEM_RESOURCE (section 2.2.12.1.7.3)** structure.

MD_Type: MUST be set to the value of the size of the **MEM_RANGE** (section 2.2.12.1.7.2) structure.

MD_Alloc_Base: For a resource list: The lowest-numbered of a range of contiguous physical memory addresses allocated to the device. For a resource requirements list: Zero.

MD_Alloc_End: For a resource list: The highest-numbered of a range of contiguous physical memory addresses allocated to the device. For a resource requirements list: Zero.

MD_Flags: A bit flag from each of the flag sets defined in the following table. Flag Definition:

Read-Only Flags

Value	Meaning
fMD_ROM 0x00000000	The specified memory range is read-only.
fMD_RAM 0x00000001	The specified memory range is not read-only.

The bitmask for the bit within MD_Flags that specifies the read-only attribute is 0x00000001 (mMD_MemoryType).

Write-Only Flags

Value	Meaning
fMD_ReadDisallowed 0x00000008	The specified memory range is write only.
fMD_ReadAllowed 0x00000000	The specified memory range is not write only.

The bitmask for the bit within MD_Flags that specifies the write-only attribute is 0x00000008 (mMD_Readable).

Address Size Flags

Value	Meaning
fMD_24 0x00000000	24-bit addressing (not used).
fMD_32 0x00000002	32-bit addressing.

The bitmask for the bit within MD_Flags that specifies the address size is 0x00000002 (mMD_32_24).

Prefetch Flags

Value	Meaning
fMD_PrefetchAllowed 0x00000004	The specified memory range can be prefetched.
fMD_PrefetchDisallowed 0x00000000	The specified memory range cannot be prefetched.

The bitmask for the bit within MD_Flags that specifies the prefetch capability is 0x00000004 (mMD_Prefetchable).

Caching Flags

Value	Meaning
fMD_Cacheable 0x00000020	The specified memory range can be cached.
fMD_NonCacheable 0x00000000	The specified memory range cannot be cached.

The bitmask for the bit within MD_Flags that specifies the caching capability is 0x00000020 (mMD_Cacheable).

Combined-Write Caching Flags

Value	Meaning
fMD_CombinedWriteAllowed	Combined write caching is allowed.

Value	Meaning
0x00000010	
fMD_CombinedWriteDisallowed 0x00000000	Combined write caching is not allowed.

The bitmask for the bit within MD_Flags that specifies the combined write caching capability is 0x00000010 (mMD_CombinedWrite).

Reserved: For internal use only.

2.2.12.1.7.2 MEM_RANGE

The **MEM_RANGE** structure specifies a resource requirements list that describes memory usage for a device instance.

```
typedef struct _MEM_RANGE {
  __int64 MR_Align;
  unsigned long MR_nBytes;
  __int64 MR_Min;
  __int64 MR_Max;
  DWORD MR_Flags;
  DWORD MR_Reserved;
} MEM_RANGE,
 *PMEM_RANGE;
```

MR_Align: Mask used to specify the memory address boundary on which the first allocated memory address must be aligned.

MR_nBytes: The number of bytes of memory required by the device.

MR_Min: The lowest numbered of a range of contiguous memory addresses that can be allocated to the device.

MR_Max: The highest numbered of a range of contiguous memory addresses that can be allocated to the device.

MR_Flags: A bit flag from each of the flag sets defined in the table included with the description of the MD_Flags member of the **MEM_DES (section 2.2.12.1.7.1)** structure.

MR_Reserved: For internal use only.

2.2.12.1.7.3 MEM_RESOURCE

The **MEM_RESOURCE** structure is used for specifying either a resource list or a resource requirements list that describes memory usage for a device instance.

```
typedef struct _MEM_RESOURCE {
  MEM_DES MEM_Header;
  MEM_RANGE MEM_Data[1];
} MEM_RESOURCE,
 *PMEM_RESOURCE;
```

MEM_Header: A **MEM_DES (section 2.2.12.1.7.1)** structure.

MEM_Data: For a resource list: Zero; For a resource requirements list: A **MEM_RANGE (section 2.2.12.1.7.2)** array.

2.2.12.1.8 MfCard Structures

The MfCard structures correspond to the ResType_MfCardConfig resource type, as specified in section 3.1.4.44.

2.2.12.1.8.1 MFCARD_DES

The **MFCARD_DES** structure is used for specifying either a resource list or a resource requirements list that describes resource usage by one of the hardware functions provided by an instance of a multifunction device.

```
typedef struct _MFCARD_DES {
  DWORD PMF_Count;
  DWORD PMF_Type;
  DWORD PMF_Flags;
  byte PMF_ConfigOptions;
  byte PMF_IoResourceIndex;
  byte PMF_Reserved[2];
  DWORD PMF_ConfigRegisterBase;
} MFCARD_DES,
 *PMFCARD_DES;
```

PMF_Count: MUST be 1.

PMF_Type: Not used; MUST be set to 0.

PMF_Flags: A bit flag is defined, as specified in the following table.

Value	Meaning
fPMF_AUDIO_ENABLE 0x00000008	If set, audio is enabled.

PMF_ConfigOptions: Contents of the 8-bit PCMCIA configuration option register.

PMF_IoResourceIndex: Zero-based index indicating the **IO_RESOURCE (section 2.2.12.1.5.3)** structure that describes the I/O resources for the hardware function being described by this **MFCARD_DES** structure.

PMF_Reserved: Not used; MUST be set to 0.

PMF_ConfigRegisterBase: Offset from the beginning of the card's attribute memory space to the base configuration register address.

2.2.12.1.8.2 MFCARD_RESOURCE

The **MFCARD_RESOURCE** structure is used for specifying either a resource list or a resource requirements list that describes resource usage by one of the hardware functions provided by an instance of a multifunction device.

```
typedef struct _MFCARD_RESOURCE {
  MFCARD_DES MfCard_Header;
} MFCARD_RESOURCE,
 *PMFCARD_RESOURCE;
```

MfCard_Header: An **MFCARD_DES (section 2.2.12.1.8.1)** structure.

2.2.12.1.9 PcCard Structures

The PcCard structures correspond to the ResType_PcCardConfig resource type, as specified in section 3.1.4.44.

2.2.12.1.9.1 PCCARD_DES

The **PCCARD_DES** structure is used for specifying either a resource list or a resource requirements list that describes resource usage by a PC card instance.

```
typedef struct _PCCARD_DES {
  DWORD PCD_Count;
  DWORD PCD_Type;
  DWORD PCD_Flags;
  byte PCD_ConfigIndex;
  byte PCD_Reserved[3];
  DWORD PCD_MemoryCardBase1;
  DWORD PCD_MemoryCardBase2;
} PCCARD_DES,
 *PPCCARD_DES;
```

PCD_Count: MUST be 1.

PCD_Type: Not used; MUST be set to 0.

PCD_Flags: A bit flag from each of the flag sets defined in the following tables.

I/O Addressing Flags

Value	Meaning
fPCD_IO_8 0x00000000	The device uses 8-bit I/O addressing.
fPCD_IO_16 0x00000001	The device uses 16-bit I/O addressing.

The bitmask for the bit within PCD_Flags that specifies 8-bit or 16-bit I/O addressing is 0x00000001 (mPCD_IO_8_16).

Memory Addressing Flags

Value	Meaning
fPCD_MEM_8 0x00000000	The device uses 8-bit memory addressing.
fPCD_MEM_16 0x00000002	The device uses 16-bit I/O addressing.

The bitmask for the bit within PCD_Flags that specifies 8-bit or 16-bit memory addressing is 0x0000000 (mPCD_MEM_8_16).

PCD_ConfigIndex: The 8-bit index value used to locate the device's configuration.

PCD_Reserved: Not used; MUST be set to 0.

PCD_MemoryCardBase1: Optional; card base address of the first memory window.

PCD_MemoryCardBase2: Optional; card base address of the second memory window.

2.2.12.1.9.2 PCCARD_RESOURCE

The **PCCARD_RESOURCE** structure is used for specifying either a resource list or a resource requirements list that describes resource usage by a PC card instance.

```
typedef struct _PCCARD_RESOURCE {
  PCCARD_DES PcCard_Header;
} PCCARD_RESOURCE,
 *PPCCARD_RESOURCE;
```

PcCard_Header: A **PCCARD_DES (section 2.2.12.1.9.1)** structure.

2.2.12.1.10 Resource Conflict Detection Structures

These structures are used by **PNP_QueryResConfList**.

2.2.12.1.10.1 PNP_CONFLICT_LIST

A **PNP_CONFLICT_LIST** structure describes a list of conflicts. The structure is a header, to be followed by an array of **PNP_CONFLICT_ENTRY (section 2.2.12.1.10.2)** structures, followed immediately by a **PNP_CONFLICT_STRINGS (section 2.2.12.1.10.3)** structure.

```
typedef struct _PNP_CONFLICT_LIST {
  unsigned long Reserved1;
  unsigned long Reserved2;
  unsigned long ConflictsCounted;
  unsigned long ConflictsListed;
  unsigned long RequiredBufferSize;
  PNP_CONFLICT_ENTRY ConflictEntry[1];
} PNP_CONFLICT_LIST,
 *PPNP_CONFLICT_LIST;
```

Reserved1: MUST be ignored on receipt.**Note** MUST be set to 0.

Reserved2: MUST be ignored on receipt.**Note** MUST be set to 0.

ConflictsCounted: The number of conflicts that have been determined.

ConflictsListed: The number of conflicts in this list.

RequiredBufferSize: The buffer size required to report all conflicts.

ConflictEntry: An array of ConflictsListed **PNP_CONFLICT_ENTRY** (section 2.2.12.1.10.2) entries.

2.2.12.1.10.2 PNP_CONFLICT_ENTRY

A **PNP_CONFLICT_ENTRY** structure describes a single conflict entry in a **PNP_CONFLICT_LIST**.

```
typedef struct _PNP_CONFLICT_ENTRY {
  unsigned long DeviceInstance;
  unsigned long DeviceFlags;
  unsigned long ResourceType;
  __int64 ResourceStart;
  __int64 ResourceEnd;
  unsigned long ResourceFlags;
} PNP_CONFLICT_ENTRY,
 *PPNP_CONFLICT_ENTRY;
```

DeviceInstance: Byte offset from start of the buffer to NULL-terminated string for device instance ID string in DeviceInstanceStrings.

DeviceFlags: Flags regarding the device. Possible values:

Value	Meaning
PNP_CE_LEGACY_DRIVER 0x00000001	DeviceInstance reports back legacy driver name.
PNP_CE_ROOT_OWNED 0x00000002	Root-owned device.
PNP_CE_TRANSLATE_FAILED 0x00000004	Translation of resource failed; resource range not available for use.

ResourceType: Type of range that conflict is with.

ResourceStart: Start of conflicting address-range.

ResourceEnd: End of conflicting address-range.

ResourceFlags: Flags regarding the conflicting resource; none defined.

2.2.12.1.10.3 PNP_CONFLICT_STRINGS

A **PNP_CONFLICT_STRINGS** structure describes conflicting devices.

```
typedef struct _PNP_CONFLICT_STRINGS {
  unsigned long NullDeviceInstance;
  wchar_t DeviceInstanceStrings[1];
} PNP_CONFLICT_STRINGS,
 *PPNP_CONFLICT_STRINGS;
```

NullDeviceInstance: Exists immediately after ConflictsListed * PNP_CONFLICT_ENTRY.**Note** MUST be 0xFFFFFFFF.

DeviceInstanceStrings: First device instance ID string.

2.2.12.2 DEVPROPKEY

The **DEVPROPKEY** structure uniquely describes a device property.

```
typedef struct _DEVPROPKEY {
  GUID fmtid;
  unsigned long pid;
} DEVPROPKEY;
```

fmtid: A globally unique identifier (GUID).

pid: Property identifier. This value MUST be greater than 2.

2.2.12.3 HWPROFILEINFO

The **HWPROFILEINFO** structure defines hardware profile information.

```
typedef struct _HWPROFILEINFO {
  unsigned long HWPI_ulHWProfile;
  wchar_t HWPI_szFriendlyName[80];
  DWORD HWPI_dwFlags;
} HWPROFILEINFO;
```

HWPI_ulHWProfile: The handle of the hardware profile.

HWPI_szFriendlyName: The friendly name of the hardware profile.

HWPI_dwFlags: Profile flags; MUST be one of the following:

Value	Meaning
CM_HWPI_NOT_DOCKABLE 0x00000000	Machine is not dockable.
CM_HWPI_UNDOCKED 0x00000001	Hardware profile is for a docked configuration.
CM_HWPI_DOCKED 0x00000002	Hardware profile is for an undocked configuration.

The **HWPROFILEINFO** structure is defined for Unicode strings. For non-Unicode implementations, the structure should define the **wchar_t** data type member as a char data type.

3 Protocol Details

The client side of this protocol is simply a pass through. That is, there are no additional timers or other state required on the client side of this protocol. Calls made by the higher-layer protocol or application are passed directly to the transport, and the results returned by the transport are passed directly back to the higher-layer protocol or application.

3.1 Server Details

The server responds to messages it receives from the client.

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to explain how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Note that the above conceptual data can be implemented using a variety of techniques. Any data structure that stores the above conceptual data may be used in the implementation.

A server implementing this RPC interface manages the state and attributes of devices and other abstract objects on the system. The server maintains a directory for storage and retrieval of the state and attributes of the devices and other abstract objects that it manages.

Each device is uniquely represented on the system by a string identifier. A server responds to queries from a client to retrieve the device instance ID strings for all (or some subset of) devices on the remote system.

A server responds to queries from a client to retrieve information on the attributes, state, or **hardware resources** of a specific device. The client may also make requests to modify the attributes, state, or hardware resources of one or more devices on the remote system. The client may also make requests to manually create devices on the server, or to remove all state for a device from the server.

A server also manages state and attributes for other abstract objects such as Device Setup Classes, device interfaces, and **Device Interface Classes**. Each of these objects is also uniquely represented on the system by a string identifier. In the case of Device Setup Classes and Device Interface Classes, the objects may also be described by a GUID. The server also responds to requests to query or modify the attributes or state of those abstract objects.

3.1.2 Timers

None.

3.1.3 Initialization

At initialization time, the Plug and Play Remote (PNPR) Protocol server MUST register the RPC interface and begin listening on the RPC well-known endpoint specified in section 2.1. The server then MUST wait for client requests.<6>

3.1.4 Message Processing Events and Sequencing Rules

This protocol MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 6.0, as specified in [MS-RPCE] section 3.<7> The methods MUST NOT throw an exception.

The server SHOULD enforce appropriate security measures to make sure the caller has required permissions to execute the following routines.<8>

Methods in RPC Opnum Order

Method	Description
Opnum00NotUsedOnWire	Reserved for local use. Opnum: 0
Opnum01NotUsedOnWire	Reserved for local use. Opnum: 1
PNP_GetVersion	Returns a static server version number, 0x0400, for the server-side component. Opnum: 2
PNP_GetGlobalState	Returns the current global state of the configuration manager. Opnum: 3
Opnum04NotUsedOnWire	Reserved for local use. Opnum: 4
Opnum05NotUsedOnWire	Reserved for local use. Opnum: 5
PNP_ValidateDeviceInstance	Verifies that the specified device instance ID represents a valid device on the server. Validation is performed by the server. Opnum: 6
PNP_GetRootDeviceInstance	Returns the device instance ID string for the device that represents the root of the hardware tree. Opnum: 7
PNP_GetRelatedDeviceInstance	Returns a device instance that is related to the specified device instance. Opnum: 8
PNP_EnumerateSubKeys	Provides generic enumeration of enumerator IDs and Device Setup Class GUIDs. Opnum: 9
PNP_GetDeviceList	Returns a list of device instances. Opnum: 10
PNP_GetDeviceListSize	Returns the size of a list of device instances. Opnum: 11

Method	Description
PNP_GetDepth	Returns the depth of a device instance in the hardware tree. Opnum: 12
PNP_GetDeviceRegProp	Returns the device's well-known properties. Opnum: 13
PNP_SetDeviceRegProp	Sets the device's well-known properties. Opnum: 14
PNP_GetClassInstance	Returns a unique string identifier that represents a location where the server stores additional configuration settings for the specified device instance. Opnum: 15
PNP_CreateKey	Creates a Device Parameters storage location for the specified device, if it does not already exist. Opnum: 16
PNP_DeleteRegistryKey	Deletes the specified storage location. Opnum: 17
PNP_GetClassCount	Returns the number of valid device setup classes currently installed. Opnum: 18
PNP_GetClassName	Returns the name of the Device Setup Class represented by the globally unique identifier (GUID). Opnum: 19
PNP_DeleteClassKey	Deletes the storage location for the specified Device Setup Class. Opnum: 20
PNP_GetInterfaceDeviceAlias	Returns the device interface that is an alias for the specified Device Interface Class GUID and device interface. Opnum: 21
PNP_GetInterfaceDeviceList	Returns a MULTI_SZ list of device interface. Opnum: 22
PNP_GetInterfaceDeviceListSize	Returns the size, in chars, of a MULTI_SZ interface device list. Opnum: 23
PNP_RegisterDeviceClassAssociation	Registers a device interface for the specified device instance and Device Interface Class, and returns the device interface ID string for the device interface. Opnum: 24
PNP_UnregisterDeviceClassAssociation	Unregisters the specified device interface. Opnum: 25
PNP_GetClassRegProp	Returns the specified well-known property value for the Device Setup Class specified by the caller-provided globally

Method	Description
	unique identifier (GUID). Opnum: 26
PNP_SetClassRegProp	Sets the specified well-known value for the specified Device Setup Class. Opnum: 27
PNP_CreateDevInst	Creates the specified device instance. Opnum: 28
PNP_DeviceInstanceAction	Performs operations—for example, setup, enable, or re-enumerate—on a device instance. It handles various routines by accepting a major action value and a minor action value. Opnum: 29
PNP_GetDeviceStatus	Returns the status and problem values for the given device instance. Opnum: 30
PNP_SetDeviceProblem	Sets the specified problem information for the specified device instance. Opnum: 31
PNP_DisableDevInst	Disables the specified device instance. Opnum: 32
PNP_UninstallDevInst	Removes all configuration data and storage locations for the specified device instance. Opnum: 33
PNP_AddID	Adds a hardware ID or a compatible ID to the registry for the specified device instance. Opnum: 34
PNP_RegisterDriver	Registers a driver for a device instance and enumerates it. Opnum: 35
PNP_QueryRemove	Queries and removes a device instance. In this context, remove refers to the state of the device, not to the removal of hardware or configuration data. Opnum: 36
PNP_RequestDeviceEject	Requests that a device be ejected. In this context, eject refers to devices such as disk drives, not to media such as disks and tapes. Opnum: 37
PNP_IsDockStationPresent	Determines if a **docking station** is present. Opnum: 38
PNP_RequestEjectPC	Requests that a computer be ejected (undocked) from its docking station. Opnum: 39

Method	Description
PNP_HwProfFlags	Gets and sets device configuration flags for the specified device instance that are applicable when the system is using the specified hardware profile. Opnum: 40
PNP_GetHwProfInfo	Returns a structure of information for the specified hardware profile. Opnum: 41
PNP_AddEmptyLogConf	Adds an empty logical configuration. Opnum: 42
PNP_FreeLogConf	Frees a logical configuration. Opnum: 43
PNP_GetFirstLogConf	Finds the first logical configuration of the specified type for the specified device instance. Opnum: 44
PNP_GetNextLogConf	Finds the next logical configuration of the specified type for the specified device instance. Opnum: 45
PNP_GetLogConfPriority	Returns the priority value assigned to the specified logical configuration. Opnum: 46
PNP_AddResDes	Adds a resource descriptor to the specified logical configuration. Opnum: 47
PNP_FreeResDes	Frees a resource descriptor for the specified logical configuration. Opnum: 48
PNP_GetNextResDes	Gets the next resource descriptor in the specified logical configuration. Opnum: 49
PNP_GetResDesData	Returns the data for the specified resource descriptor. Opnum: 50
PNP_GetResDesDataSize	Returns the data size for the specified resource descriptor. Opnum: 51
PNP_ModifyResDes	Modifies the specified resource descriptor. Opnum: 52
PNP_DetectResourceConflict	Detects conflicts with the specified resource descriptor. Opnum: 53
PNP_QueryResConfList	Returns conflict information for a specified resource. Opnum: 54

Method	Description
Opnum55NotUsedOnWire	Reserved for local use. Opnum: 55
Opnum56NotUsedOnWire	Reserved for local use. Opnum: 56
Opnum57NotUsedOnWire	Reserved for local use. Opnum: 57
Opnum58NotUsedOnWire	Reserved for local use. Opnum: 58
Opnum59NotUsedOnWire	Reserved for local use. Opnum: 59
Opnum60NotUsedOnWire	Reserved for local use. Opnum: 60
PNP_GetCustomDevProp	Returns the data for a device instance custom property. Opnum: 61
PNP_GetVersionInternal	Returns the internal version number for the server-side component. Opnum: 62
PNP_GetBlockedDriverInfo	Returns the list of drivers that have been blocked from loading on the system since the system was booted. Opnum: 63
PNP_GetServerSideDeviceInstallFlags	Returns the server-side device installation flags. Opnum: 64
PNP_GetObjectPropKeys	Received by the server in an RPC_REQUEST packet. In response, the server returns the properties currently set on the specified object. Opnum: 65
PNP_GetObjectProp	Received by the server in an RPC_REQUEST packet. In response, the server returns the specified object property. Opnum: 66
PNP_SetObjectProp	Received by the server in an RPC_REQUEST packet. In response, the server sets the specified object property. Opnum: 67

In the table above, the term "Reserved for local use" means that the client MUST NOT send the opnum, and the server behavior is undefined <9>since it does not affect interoperability.

3.1.4.1 PNP_GetVersion (Opnum 2)

The **PNP_GetVersion** method returns a static server version number, 0x0400, for the server-side component. Use the **PNP_GetVersionInternal (section 3.1.4.53)** method to obtain the real internal version of the server.<10>

```
DWORD PNP_GetVersion(
  [in] handle_t hBinding,
  [out] unsigned short* pVersion
);
```

hBinding: A remote procedure call (RPC) binding handle.

pVersion: Returns a pointer to the static version number. This value is always 0x0400.

Return Values: The method returns one of the following codes:

Return value/code	Description
0x00000000 CR_SUCCESS	The version was successfully returned.
0x00000013 CR_FAILURE	General failure.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.2 PNP_GetGlobalState (Opnum 3)

The **PNP_GetGlobalState** method returns the current global state of the configuration manager.<11>

```
DWORD PNP_GetGlobalState(
  [in] handle_t hBinding,
  [out] unsigned long* pulState,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pulState: The current global state. Possible values:

Value	Meaning
CM_GLOBAL_STATE_CAN_DO_UI 0x00000001	CM can display the user interface (UI).
CM_GLOBAL_STATE_SERVICES_AVAILABLE 0x00000004	CM APIs are available.
CM_GLOBAL_STATE_SHUTTING_DOWN 0x00000008	CM is shutting down.
CM_GLOBAL_STATE_DETECTION_PENDING 0x00000010	State detection is pending.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has the value 0x00000000.

Return value/code	Description
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000013 CR_FAILURE	General failure.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.3 PNP_ValidateDeviceInstance (Opnum 6)

The **PNP_ValidateDeviceInstance** method verifies that the specified device instance ID represents a valid device on the server. Validation is performed by the server.<12>

```
DWORD PNP_ValidateDeviceInstance(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that contains a device instance ID string for the device to be validated.

ulFlags: A CM_LOCATE_DEVNODE_* device validation flag value. Possible values:

Value	Meaning
CM_LOCATE_DEVNODE_NORMAL 0x00000000	Validate only device instances that are currently active on the server.
CM_LOCATE_DEVNODE_PHANTOM 0x00000001	Allow validation of a device instance that is not currently active, but that does exist on the server.
CM_LOCATE_DEVNODE_CANCELREMOVE 0x00000002	If the device is marked for pending removal, cancel the removal.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.

Return value/code	Description
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000013 CR_FAILURE	General failure.
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not exist, or does not correspond to a present device.
0x00000033 ACCESS_DENIED	Verification of client access privileges failed.

3.1.4.4 PNP_GetRootDeviceInstance (Opnum 7)

The **PNP_GetRootDeviceInstance** method returns the device instance ID string for the device that represents the root of the hardware tree.<13>

```
DWORD PNP_GetRootDeviceInstance(
  [in] handle_t hBinding,
  [out, string, size_is(ulLength)]
    wchar_t* pDeviceID,
  [in] PNP_RPC_STRING_LEN ulLength
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Pointer to a buffer that receives the root device instance string.<14>

ulLength: Size in characters of the *pDeviceID* buffer.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:**Note** CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path

Return value/code	Description
	components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available.
0x0000001D CR_REGISTRY_ERROR	An error occurred while trying to create the root device instance in the registry.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.5 PNP_GetRelatedDeviceInstance (Opnum 8)

The **PNP_GetRelatedDeviceInstance** method returns a device instance that is related to the specified device instance.<15>

```
DWORD PNP_GetRelatedDeviceInstance(
  [in] handle_t hBinding,
  [in] unsigned long ulRelationship,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [out, string, size_is(*pulLength)]
    wchar_t* pRelatedDeviceID,
  [in, out] PNP_RPC_STRING_LEN* pulLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

ulRelationship: Specifies the relationship of the device instance to be retrieved. Possible values:

Value	Meaning
PNP_GET_PARENT_DEVICE_INSTANCE 0x00000001	Parent device.
PNP_GET_CHILD_DEVICE_INSTANCE 0x00000002	Child device.
PNP_GET_SIBLING_DEVICE_INSTANCE 0x00000003	Sibling device.

pDeviceID: Null-terminated string that contains a device instance ID string for the device to retrieve the relation for.

pRelatedDeviceID: Pointer to a buffer that receives the related device instance ID string.

pulLength: Length in characters of the *pRelatedDeviceID* buffer.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	The specified *pulLength* or *pRelatedDeviceID* parameter value is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available.
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not exist, or no devices match the specified relation.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.6 PNP_EnumerateSubKeys (Opnum 9)

The **PNP_EnumerateSubKeys** method provides generic enumeration of enumerator IDs and Device Setup Class GUIDs.<16>

```
DWORD PNP_EnumerateSubKeys(
  [in] handle_t hBinding,
  [in] unsigned long ulBranch,
  [in] unsigned long ulIndex,
  [out, string, size_is(ulLength)]
```

```
    wchar_t* Buffer,
  [in] PNP_RPC_STRING_LEN ulLength,
  [out] PNP_RPC_STRING_LEN* pulRequiredLen,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

ulBranch: Specifies what type of identifier strings to enumerate. Possible values:

Value	Meaning
PNP_ENUMERATOR_SUBKEYS 0x00000001	Enumerate enumerator ID strings.
PNP_CLASS_SUBKEYS 0x00000002	Enumerate Device Setup ClassGUID strings.

ulIndex: Index of the element to retrieve.

Buffer: Returns the name of the enumerated element.

ulLength: Maximum size in characters of Buffer.

pulRequiredLen: If the transfer is successful, contains the number of characters actually copied to *Buffer*. If the buffer is too small, contains the number of characters required.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	The *pulRequiredLen* or *Buffer* parameter value is invalid.
0x00000004 CR_INVALID_FLAG	The *ulFlags* parameter value is invalid.
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	The output buffer, *Buffer*, is too small to hold all the data available.
0x0000001D CR_REGISTRY_ERROR	An error occurred on an attempt to read the registry.
0x00000025 CR_NO_SUCH_VALUE	No such value exists.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.7 PNP_GetDeviceList (Opnum 10)

The **PNP_GetDeviceList** method returns a list of device instances.

```
DWORD PNP_GetDeviceList(
  [in] handle_t hBinding,
  [in, string, unique, range(0,PNP_MAX_STRING_LEN)]
    wchar_t* pszFilter,
  [out, size_is(*pulLength), length_is(*pulLength)]
    wchar_t* Buffer,
  [in, out] PNP_RPC_BUFFER_SIZE* pulLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszFilter: Optional null-terminated string parameter that, if specified, limits the list to device instances returned.

The value of the optional *pszFilter* parameter can depend on the device ID list flag values specified by *ulFlags*.

If *ulFlags* specifies the CM_GETIDLIST_FILTER_ENUMERATOR flag, *pszFilter* MAY be NULL or MUST specify a NULL-terminated string that contains either an Enumerator ID or a device ID.

If *ulFlags* specifies the CM_GETIDLIST_FILTER_SERVICE flag, *pszFilter* MUST specify a NULL-terminated string that represents the name of a service.

Buffer: Pointer to a buffer containing the MULTI_SZ list of device instance ID strings. In a MULTI_SZ buffer, each string is null-terminated, as is the set of strings, making the last string doubly null-terminated.

pulLength: On input, contains the size in chars of *Buffer* transferred on output.

ulFlags: Device ID list flag specifying what device IDs to return. Possible values:

Value	Meaning
CM_GETIDLIST_FILTER_NONE 0x00000000	Optional *pszFilter* parameter is ignored, and a list of all devices on the system is returned.
CM_GETIDLIST_FILTER_ENUMERATOR 0x00000001	The *pszFilter* parameter contains the string for a bus enumerator ID or a device ID, and returns a list of all matching device instances.
CM_GETIDLIST_FILTER_SERVICE 0x00000002	The *pszFilter* parameter contains the string for a service; for example, USBSTOR or EL90XBC. The list contains all device instances in the system whose service matches *pszFilter*, as well as any devices for which *pszFilter* is listed as an UpperFilter or a LowerFilter.
CM_GETIDLIST_FILTER_EJECTRELATIONS 0x00000004	The *pszFilter* parameter is a full device instance ID. The returned list contains other device instances that have the ejection relations to the specified device instance.

Value	Meaning
CM_GETIDLIST_FILTER_REMOVALRELATIONS 0x00000008	The *pszFilter* parameter is a full device instance ID. The returned list contains device instances that have removal relations to the specified device instance.
CM_GETIDLIST_FILTER_POWERRELATIONS 0x00000010	The *pszFilter* parameter is a full device instance ID. The returned list contains device instances that have power relations to the specified device instance.
CM_GETIDLIST_FILTER_BUSRELATIONS 0x00000020	The *pszFilter* parameter is a full device instance ID. The returned list contains device instances that have bus relations to the specified device instance.
CM_GETIDLIST_DONOTGENERATE 0x10000040	Can be specified with the CM_GETIDLIST_FILTER_SERVICE flag to prevent the system from creating a new legacy device instance for the service.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *pulLength* or *pszFilter* parameter value may be invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	The output buffer, Buffer, is too small to hold all the data available.
0x0000001D CR_REGISTRY_ERROR	An error occurred during an attempt to read the registry.
0x0000001E CR_INVALID_DEVICE_ID	The specified device ID is invalid.
0x00000025 CR_NO_SUCH_VALUE	No such value exists.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.8 PNP_GetDeviceListSize (Opnum 11)

The **PNP_GetDeviceListSize** method returns the size of a list of device instances.

```
DWORD PNP_GetDeviceListSize(
  [in] handle_t hBinding,
  [in, string, unique, range(0,PNP_MAX_STRING_LEN)]
    wchar_t* pszFilter,
  [out] PNP_RPC_BUFFER_SIZE* pulLen,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszFilter: Optional null-terminated string parameter that, if specified, limits the list size to device instances returned.

The value of the optional *pszFilter* parameter can depend on the device ID list flag values specified by *ulFlags*.

If *ulFlags* specifies the CM_GETIDLIST_FILTER_ENUMERATOR flag, *pszFilter* MAY be NULL or MUST specify a NULL-terminated string that contains either an enumerator ID or a device ID.

If *ulFlags* specifies the CM_GETIDLIST_FILTER_SERVICE flag, *pszFilter* MUST specify a NULL-terminated string that represents the name of a service.

pulLen: Returns the estimate of the largest possible size in characters of a device instance list.

ulFlags: Flag specifying what device IDs to return. For possible values, see section 3.1.4.7.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *pulLen* or *pszFilter* parameter value may be invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available.
0x0000001D CR_REGISTRY_ERROR	An error occurred during an attempt to read the registry.
0x0000001E CR_INVALID_DEVICE_ID	The specified device ID is invalid.
0x00000025 CR_NO_SUCH_VALUE	No such value exists.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.9 PNP_GetDepth (Opnum 12)

The **PNP_GetDepth** method returns the depth of a device instance in the hardware tree.<17>

```
DWORD PNP_GetDepth(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pszDeviceID,
  [out] unsigned long* pulDepth,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszDeviceID: Null-terminated string that contains a device instance ID string for the device for which to find the depth in the hardware tree.

pulDepth: Returns a pointer to the depth of pszDeviceID. This value is 0 to designate the root of the tree or 1 to designate a child of the root, and continues to increment for each child tier.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	The specified *pulDepth* parameter value is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000005 CR_INVALID_DEVNODE	The specified *pszDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000013 CR_FAILURE	General failure.
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.

Return value/code	Description
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.10 PNP_GetDeviceRegProp (Opnum 13)

The **PNP_GetDeviceRegProp** method returns the device's well-known properties.<18>

```
DWORD PNP_GetDeviceRegProp(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in] unsigned long ulProperty,
  [in, out] unsigned long* pulRegDataType,
  [out, size_is(*pulTransferLen), length_is(*pulTransferLen)]
    byte far* Buffer,
  [in, out] PNP_PROP_SIZE* pulTransferLen,
  [in, out] PNP_PROP_SIZE* pulLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that contains a device instance ID string for the device for which the property is returned.

ulProperty: Identifier specifying what well-known registry property value to get. Possible values:

Value	Meaning
CM_DRP_DEVICEDESC 0x00000001	DeviceDesc REG_SZ property (read/write).
CM_DRP_HARDWAREID 0x00000002	HardwareID REG_MULTI_SZ property (read/write).
CM_DRP_COMPATIBLEIDS 0x00000003	CompatibleIDs REG_MULTI_SZ property (read/write).
CM_DRP_SERVICE 0x00000005	Service REG_SZ property (read/write).
CM_DRP_CLASS 0x00000008	Class REG_SZ property (read/write).
CM_DRP_CLASSGUID 0x00000009	Class GUID REG_SZ property (read/write).
CM_DRP_DRIVER 0x0000000A	Driver REG_SZ property (read/write).

Value	Meaning
CM_DRP_CONFIGFLAGS 0x0000000B	ConfigFlags REG_DWORD property (read/write).
CM_DRP_MFG 0x0000000C	Manufacturer REG_SZ property (read/write).
CM_DRP_FRIENDLYNAME 0x0000000D	FriendlyName REG_SZ property (read/write).
CM_DRP_LOCATION_INFORMATION 0x0000000E	LocationInformation REG_SZ property (read/write).
CM_DRP_PHYSICAL_DEVICE_OBJECT_NAME 0x0000000F	PhysicalDeviceObjectName REG_SZ property (read-only).
CM_DRP_CAPABILITIES 0x00000010	Capabilities REG_DWORD property (read-only).
CM_DRP_UI_NUMBER 0x00000011	User interface number REG_DWORD property (read-only).
CM_DRP_UPPERFILTERS 0x00000012	UpperFilters REG_MULTI_SZ property (read/write).
CM_DRP_LOWERFILTERS 0x00000013	LowerFilters REG_MULTI_SZ property (read/write).
CM_DRP_BUSTYPEGUID 0x00000014	Bus type GUID property (read-only).
CM_DRP_LEGACYBUSTYPE 0x00000015	Legacy bus type INTERFACE_TYPE property (read-only).
CM_DRP_BUSNUMBER 0x00000016	Bus number DWORD property (read-only).
CM_DRP_ENUMERATOR_NAME 0x00000017	Enumerator name REG_SZ property (read-only).
CM_DRP_SECURITY 0x00000018	Device security descriptor override REG_BINARY property (read/write).
CM_DRP_SECURITY_SDS 0x00000019	Device security descriptor override REG_BINARY property (read/write). Specifies a binary security descriptor in the self-relative format to override the default security settings applied to the device.
CM_DRP_DEVTYPE 0x0000001A	Device type override REG_DWORD property (read/write).
CM_DRP_EXCLUSIVE 0x0000001B	Device exclusivity override REG_DWORD property (read/write).
CM_DRP_CHARACTERISTICS 0x0000001C	Device characteristics override REG_DWORD property (read/write).
CM_DRP_ADDRESS 0x0000001D	Device address REG_DWORD property (read-only).

Value	Meaning
CM_DRP_UI_NUMBER_DESC_FORMAT 0x0000001E	User interface number description format REG_SZ property (read/write).
CM_DRP_DEVICE_POWER_DATA 0x0000001F	Device power data REG_BINARY property (read-only).
CM_DRP_REMOVAL_POLICY 0x00000020	Device removal policy REG_DWORD property (read-only).
CM_DRP_REMOVAL_POLICY_HW_DEFAULT 0x00000021	Device removal policy (hardware default setting) REG_DWORD property (read-only).
CM_DRP_REMOVAL_POLICY_OVERRIDE 0x00000022	Device removal policy (override setting) REG_DWORD property (read/write).
CM_DRP_INSTALL_STATE 0x00000023	Device installation state REG_DWORD property (read-only).
CM_DRP_LOCATION_PATHS 0x00000024	CM_DRP_LOCATION_PATHS: Device location paths REG_MULTI_SZ property (read-only).

pulRegDataType: Registry data type for this property; for example, REG_SZ. Possible values are predefined registry types:

Value	Meaning
REG_NONE 0	No value type is defined.
REG_SZ 1	Null-terminated string; either Unicode or ANSI, depending on which set of functions is used.
REG_EXPAND_SZ 2	Null-terminated string that contains unexpanded references to environment variables; for example, %PATH%. It is either a Unicode or an ANSI string, depending on which set of functions is used.
REG_BINARY 3	Binary data in any form.
REG_DWORD 4	A 32-bit number.
REG_DWORD_LITTLE_ENDIAN 4	A 32-bit number in little-endian format; equivalent to REG_DWORD.
REG_DWORD_BIG_ENDIAN 5	A 32-bit number in big-endian format.
REG_LINK 6	Unicode symbolic link.
REG_MULTI_SZ 7	Array of null-terminated strings that are terminated by two NULL characters.
REG_RESOURCE_LIST 8	A device driver's list of hardware resources used by the driver or one of the physical devices it controls.

Value	Meaning
REG_FULL_RESOURCE_DESCRIPTOR 9	A list of hardware resources that a physical device is using.
REG_RESOURCE_REQUIREMENTS_LIST 10	A device driver's list of possible hardware resources that it (or one of the physical devices it controls) can use.
REG_QWORD 11	A 64-bit number.
REG_QWORD_LITTLE_ENDIAN 11	A 64-bit number in little-endian format; equivalent to REG_QWORD.

Buffer: Returns the registry data. MUST NOT be NULL.

pulTransferLen: Pointer to data that indicates how many bytes to copy back into the user buffer. The *pulTransferLen* parameter is used to control how much data is marshaled by RPC between address spaces. The *pulTransferLen* parameter should be set on entry to the size of *Buffer*.

pulLength: Pointer parameter passed in by caller. On entry, it contains the size of the buffer in bytes. On exit, it contains either the amount of data copied to the caller's buffer if a transfer occurred, or the size of the buffer required to hold the property data if the buffer is too small.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *pulTransferLen* or *pulLength* parameter value may be invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x0000001A CR_BUFFER_SMALL	The *pulLength* output parameter is too small to hold all the data available.
0x0000001D	An error occurred during an attempt to read the registry.

Return value/code	Description
CR_REGISTRY_ERROR	
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.
0x00000025 CR_NO_SUCH_VALUE	The specified property value does not exist.
0x00000035 CR_INVALID_PROPERTY	The specified *ulProperty* property value is invalid.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.11 PNP_SetDeviceRegProp (Opnum 14)

The **PNP_SetDeviceRegProp** method sets the device's well-known properties.<19>

```
DWORD PNP_SetDeviceRegProp(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in] unsigned long ulProperty,
  [in] unsigned long ulDataType,
  [in, size_is(ulLength)] byte far* Buffer,
  [in] PNP_PROP_SIZE ulLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that contains a device instance ID string for the device for which the property is set.

ulProperty: Identifier specifying what registry property value to set. For a full description and list of possible values, see **PNP_GetDeviceRegProp (Opnum 13)**.

ulDataType: Registry data type for the property; for example, REG_SZ. For possible registry data type values, see **PNP_GetDeviceRegProp (Opnum 13)**.

Buffer: Registry data to be written. MUST NOT be NULL.

ulLength: Passed in by the caller. On entry, it contains the size of the buffer in bytes. MUST be set to 0 to delete the specified property.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available. The *ulLength* parameter may be too small.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.
0x00000025 CR_NO_SUCH_VALUE	The specified property value does not exist.
0x00000028 CR_NOT_DISABLEABLE	The device cannot be disabled.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.
0x00000035 CR_INVALID_PROPERTY	The specified property type, *ulProperty*, is invalid for this operation.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.12 PNP_GetClassInstance (Opnum 15)

The **PNP_GetClassInstance** method returns a unique string identifier that represents a location where the server stores additional configuration settings for the specified device instance.

```
DWORD PNP_GetClassInstance(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
```

```
  [out, string, size_is(ulLength)]
    wchar_t* pszClassInstance,
  [in] PNP_RPC_STRING_LEN ulLength
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that contains a device instance ID string for the device for which the class instance string is returned.

pszClassInstance: Pointer to a buffer that receives the null-terminated class instance string.

A class instance string is a unique identifier that represents a location where the server stores additional configuration settings for this device instance. This identifier MUST be unique among all devices on the system, and MUST be the same as the data retrieved or set as the CM_DRP_DRIVER well-known registry property for the device. See the registry property values in sections 3.1.4.10 and 3.1.4.23.<20>

ulLength: Specifies the size in characters of the *pszClassInstance* buffer.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *ulLength* parameter value may be invalid.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available. The *ulLength* parameter may be too small.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000025 CR_NO_SUCH_VALUE	The specified property value does not exist.
0x00000033	Verification of client access privileges failed.

Return value/code	Description
CR_ACCESS_DENIED	

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.13 PNP_CreateKey (Opnum 16)

The **PNP_CreateKey** method creates a device parameters storage location for the specified device, if it does not already exist.<21>

```
DWORD PNP_CreateKey(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_CM_PATH)]
    wchar_t* pszSubKey,
  [in] DWORD samDesired,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszSubKey: device instance ID string of the device to create a storage location for.

samDesired: Not used. MUST be set to 0.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Not enough memory is available to process this command.
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *pszSubKey* parameter value may be invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000005 CR_INVALID_DEVNODE	The specified *pszSubKey* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C

Return value/code	Description
	(',')
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.14 PNP_DeleteRegistryKey (Opnum 17)

The **PNP_DeleteRegistryKey** method deletes the specified storage location.<22>

```
DWORD PNP_DeleteRegistryKey(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pszDeviceID,
  [in, string, ref, range(0,PNP_MAX_CM_PATH)]
    wchar_t* pszParentKey,
  [in, string, ref, range(0,PNP_MAX_CM_PATH)]
    wchar_t* pszChildKey,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszDeviceID: Null-terminated string that contains a device instance ID string for the device to delete the storage location for.

pszParentKey: Null-terminated string that specifies the parent path of the device-specific storage location to be deleted.

pszChildKey: Null-terminated string that specifies the device-specific storage location to be deleted. The *pszChildKey* parameter may be a single path or a compound path.

ulFlags: Indicates whether the specified global registry storage location for the device are to be deleted, or if specified profile-specific registry locations for the device are to be deleted.

Value	Meaning
0x00000000	Specified global registry storage location for the device is deleted (but profile-specific keys are not deleted).
0xFFFFFFFF	All specified profile-specific registry locations for the device are deleted (but global keys are not deleted).

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Not enough memory is available to process this command.
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *pszDeviceID*, *pszParentKey*, or *pszChildKey* parameter value may be invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000005 CR_INVALID_DEVNODE	The specified *pszDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.15 PNP_GetClassCount (Opnum 18)

The **PNP_GetClassCount** method returns the number of valid Device Setup Classes currently installed.<23>

```
DWORD PNP_GetClassCount(
  [in] handle_t hBinding,
  [out] unsigned long* pulClassCount,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pulClassCount: Address of the variable that returns the number of classes installed.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000034 CR_CALL_NOT_IMPLEMENTED	The call is not implemented.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.16 PNP_GetClassName (Opnum 19)

The **PNP_GetClassName** method returns the name of the Device Setup Class represented by the globally unique identifier (GUID).<24>

```
DWORD PNP_GetClassName(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_GUID_STRING_LEN)]
    wchar_t* pszClassGuid,
  [out, string, size_is(*pulLength)]
    wchar_t* Buffer,
  [in, out] PNP_RPC_STRING_LEN* pulLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszClassGuid: String containing the Device Setup Class GUID for which a class name is retrieved.

Buffer: Buffer that returns the class name.

pulLength: On input, *pulLength* specifies the size in characters of the buffer pointed to by the *Buffer* parameter. On output, it contains the number of characters actually copied to the *Buffer* parameter.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *pulLength* or *Buffer* parameter value may be invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.

Return value/code	Description
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.17 PNP_DeleteClassKey (Opnum 20)

The **PNP_DeleteClassKey** method deletes the storage location for the specified Device Setup Class.<25>

```
DWORD PNP_DeleteClassKey(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_GUID_STRING_LEN)]
    wchar_t* pszClassGuid,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszClassGuid: The globally unique identifier (GUID) of the class with the storage location to be deleted.

ulFlags: Flags that specify class subkey deletion. Possible values:

Value	Meaning
CM_DELETE_CLASS_ONLY 0x00000000	Delete only the storage location for the specified Device Setup Class.
CM_DELETE_CLASS_SUBKEYS 0x00000001	Delete the storage location for the specified Device Setup Class as well as all storage locations returned by **PNP_GetClassInstance** for all device instances whose CM_DRP_CLASSGUID property matches the specified Device Setup Class.<26>

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *pszClassGuid* parameter value may be invalid.

Return value/code	Description
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.18 PNP_GetInterfaceDeviceAlias (Opnum 21)

The **PNP_GetInterfaceDeviceAlias** method returns the device interface that is an alias for the specified Device Interface Class GUID and device interface.<27>

```
DWORD PNP_GetInterfaceDeviceAlias(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVINTERFACE_LEN)]
    wchar_t* pszInterfaceDevice,
  [in] GUID* AliasInterfaceGuid,
  [out, string, size_is(*pulTransferLen)]
    wchar_t* pszAliasInterfaceDevice,
  [in, out] PNP_RPC_STRING_LEN* pulLength,
  [in, out] PNP_RPC_STRING_LEN* pulTransferLen,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszInterfaceDevice: Interface device for which to find an alias.

AliasInterfaceGuid: Device Interface Class GUID.

pszAliasInterfaceDevice: Address of a variable that returns the null-terimated device interface ID string for the device interface that is considered to be an alias of the specified device interface; that is, a member of the specified alias Device Interface Class that is registered to the same device with the same reference string component (if any).

pulLength: Parameter passed in by the client. On entry, it contains the size of the buffer in bytes. On exit, it contains either the amount of data copied to the caller's buffer if a transfer occurred, or the size required to hold the property data if the buffer is too small.

pulTransferLen: Indicates how much data to copy back into the user buffer.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *pulLength*, *AliasInterfaceGuid*, *pszInterfaceDevice*, *pszAliasInterfaceDevice*, or *pulTransferLen* parameter value may be invalid.
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	The method failed to get the device interface alias. An output parameter is too small to hold all the data available.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.19 PNP_GetInterfaceDeviceList (Opnum 22)

The **PNP_GetInterfaceDeviceList** method returns a MULTI_SZ list of device interfaces.<28>

```
DWORD PNP_GetInterfaceDeviceList(
  [in] handle_t hBinding,
  [in] GUID* InterfaceGuid,
  [in, string, unique, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pszDeviceID,
  [out, size_is(*pulLength), length_is(*pulLength)]
    wchar_t* Buffer,
  [in, out] PNP_RPC_BUFFER_SIZE* pulLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

InterfaceGuid: Device Interface Class for which the list of device interface ID strings is to be retrieved.

pszDeviceID: Optionally, specifies a null-terminated string that contains a device instance ID string. If specified, limits the list to device interfaces registered for the specified device.

Buffer: Pointer to a buffer containing the MULTI_SZ list of device interface ID strings. In a MULTI_SZ buffer, each string is null-terminated, as is the set of strings, making the last string doubly null-terminated.

pulLength: Size in characters of *Buffer*.

ulFlags: Flag specifying what device interfaces to return. Possible values:

Value	Meaning
CM_GET_DEVICE_INTERFACE_LIST_PRESENT 0x00000000	List only device interfaces that are currently active on the server.
CM_GET_DEVICE_INTERFACE_LIST_ALL_DEVICES 0x00000001	List all registered device interfaces, whether currently active on the server or not.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *InterfaceGuid*, *pulLength*, or *Buffer* parameter value may be invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000005 CR_INVALID_DEVNODE	The specified *pszDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed. Access to set the hardware profile flag value is denied.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.20 PNP_GetInterfaceDeviceListSize (Opnum 23)

The **PNP_GetInterfaceDeviceListSize** method returns the size in characters of a MULTI_SZ interface device list.<29>

```
DWORD PNP_GetInterfaceDeviceListSize(
  [in] handle_t hBinding,
  [out] PNP_RPC_BUFFER_SIZE* pulLen,
  [in] GUID* InterfaceGuid,
  [in, string, unique, range(0,PNP_MAX_DEVICE_ID_LEN)]
```

```
    wchar_t* pszDeviceID,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pulLen: Returns the size of the buffer in characters required to hold the MULTI_SZ interface device list.

InterfaceGuid: Device Interface Class for which the size of a list of device interface ID strings is to be retrieved.

pszDeviceID: Optionally, specifies a null-terminated string that contains a device instance ID string. If specified, limits the list size to device interface registered for the specified device.

ulFlags: Flag specifying what device interface to return. Possible values:

Value	Meaning
CM_GET_DEVICE_INTERFACE_LIST_PRESENT 0x00000000	List only device interface currently active on the server.
CM_GET_DEVICE_INTERFACE_LIST_ALL_DEVICES 0x00000001	List all registered device interface whether currently active on the server or not.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *InterfaceGuid* or *pulLen* parameter value may be invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000005 CR_INVALID_DEVNODE	The specified *pszDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000013 CR_FAILURE	General failure.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.21 PNP_RegisterDeviceClassAssociation (Opnum 24)

The **PNP_RegisterDeviceClassAssociation** method registers a device interface for the specified device instance and Device Interface Class, and returns the device interface ID string for the device interface.<30>

```
DWORD PNP_RegisterDeviceClassAssociation(
  [in] handle_t hBinding,
  [in, string, unique, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pszDeviceID,
  [in] GUID* InterfaceGuid,
  [in, string, unique, range(0,PNP_MAX_STRING_LEN)]
    wchar_t* pszReference,
  [out, string, size_is(*pulTransferLen)]
    wchar_t* pszSymLink,
  [in, out] PNP_RPC_STRING_LEN* pulLength,
  [in, out] PNP_RPC_STRING_LEN* pulTransferLen,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszDeviceID: Null-terminated string that contains a device instance ID string for the device to register the device interface for.

InterfaceGuid: Device Interface Class GUID.

pszReference: Optional. Null-terminated reference string ID string.

pszSymLink: Returns the null-terminated device interface ID string.

pulLength: Passed in by the caller. On entry, it contains the size of the buffer in characters. On exit, it contains either the amount of data copied to the caller's buffer if a transfer occurred, or the size of buffer required to hold the property data if the buffer is too small.

The pointer passed in as the *pulTransferLen* argument MUST NOT be the same as the pointer passed in for the *pulLength* argument.

pulTransferLen: Amount of data to copy into the user's buffer.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *InterfaceGuid*, *pszSymLink*, *pulTransferLen*, or *pulLength* parameter value may be missing, or its length may be invalid.

Return value/code	Description
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000005 CR_INVALID_DEVNODE	The specified *pszDeviceID* parameter value is invalid because the device instance string does not conform to the appropriate rules, which are: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available. The *pulLength* parameter may be too small.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.22 PNP_UnregisterDeviceClassAssociation (Opnum 25)

The **PNP_UnregisterDeviceClassAssociation** method unregisters the specified device interface.<31>

```
DWORD PNP_UnregisterDeviceClassAssociation(
  [in] handle_t hBinding,
  [in, string, unique, range(0,PNP_MAX_DEVINTERFACE_LEN)]
    wchar_t* pszInterfaceDevice,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszInterfaceDevice: Null-terminated device interface ID string that specifies the device interface to unregister.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *pszInterfaceDevice* parameter value may be invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000013 CR_FAILURE	General failure.
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.
0x00000025 CR_NO_SUCH_VALUE	The value does not exist.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.
0x00000036 CR_DEVICE_INTERFACE_ACTIVE	The device interface is active.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.23 PNP_GetClassRegProp (Opnum 26)

The **PNP_GetClassRegProp** method returns the specified well-known property value for the Device Interface Class specified by the caller-provided globally unique identifier (GUID).<32>

```
DWORD PNP_GetClassRegProp(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_GUID_STRING_LEN)]
    wchar_t* pszClassGuid,
  [in] unsigned long ulProperty,
  [in, out] unsigned long* pulRegDataType,
  [out, size_is(*pulTransferLen), length_is(*pulTransferLen)]
    byte far* Buffer,
  [in, out] PNP_PROP_SIZE* pulTransferLen,
  [in, out] PNP_PROP_SIZE* pulLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszClassGuid: String containing the Device Setup Class GUID whose property value is returned.

ulProperty: Identifier specifying what registry property value to get. Possible values:

Value	Meaning
CM_CRP_SECURITY 0x00000018	Class default security descriptor override REG_BINARY property (read/write). Specifies a binary security descriptor in the self-relative format to override the default security settings applied to all devices within the specified Device Setup Class.

Value	Meaning
CM_CRP_DEVTYPE 0x0000001A	Class default device type override REG_DWORD property (read/write).
CM_CRP_EXCUSIVE 0x0000001B	Class default device exclusivity override REG_DWORD property (read/write).
CM_CRP_CHARACTERISTICS 0x0000001C	Class default device characteristics override REG_DWORD property (read/write).

pulRegDataType: Optional parameter. Returns the registry data type for the specified property; for example, REG_SZ. For possible registry data type values, see section 3.1.4.10.

Buffer: Returns the registry data. MUST NOT be NULL.

pulTransferLen: Indicates how much data to copy back into the user buffer. The *pulTransferLen* parameter is used to control how much data is marshaled by RPC between address spaces; it should be set on entry to the size of *Buffer*.

The pointer passed in as the *pulTransferLen* argument MUST NOT be the same as the pointer passed in for the *pulLength* argument.

pulLength: Parameter passed in by caller. On entry, it contains the size in bytes of the buffer. On exit, it returns either the amount of data copied to the caller's buffer if a transfer occurred, or the size of buffer required to hold the property data if the buffer is too small.

The pointer passed in as the *pulTransferLen* argument MUST NOT be the same as the pointer passed in for the *pulLength* argument.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid. The *pulTransferLen* or *pulLength* parameter value may be invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available. The *pulLength* parameter may be too small.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000025 CR_NO_SUCH_VALUE	The specified value does not exist.
0x0000002E	The specified key does not exist in the registry.

Return value/code	Description
CR_NO_SUCH_REGISTRY_KEY	
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.
0x00000035 CR_INVALID_PROPERTY	The specified property type is invalid for this operation.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.24 PNP_SetClassRegProp (Opnum 27)

The **PNP_SetClassRegProp** method sets the specified well-known value for the specified Device Setup Class.<33>

```
DWORD PNP_SetClassRegProp(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_GUID_STRING_LEN)]
    wchar_t* pszClassGuid,
  [in] unsigned long ulProperty,
  [in] unsigned long ulDataType,
  [in, size_is(ulLength)] byte far* Buffer,
  [in] PNP_PROP_SIZE ulLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszClassGuid: String containing the Device Setup Class globally unique identifier (GUID) whose property is written to.

ulProperty: Identifier specifying what registry property value to set. For possible values, see section 3.1.4.23.

ulDataType: Registry data type for the specified property; for example, REG_SZ. For possible registry data type values, see section 3.1.4.10.

Buffer: Registry data to be written. MUST NOT be NULL.

ulLength: Passed in by the caller. On entry, it contains the size of the buffer in bytes.

Note MUST be set to 0 when deleting the specified property.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000004	The specified *ulFlags* parameter value is invalid.

Return value/code	Description
CR_INVALID_FLAG	
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x000002E CR_NO_SUCH_REGISTRY_KEY	The specified key does not exist in the registry.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.
0x00000035 CR_INVALID_PROPERTY	The specified property type is invalid for this operation.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.25 PNP_CreateDevInst (Opnum 28)

The **PNP_CreateDevInst** method creates the specified device instance.<34>

```
DWORD PNP_CreateDevInst(
  [in] handle_t hBinding,
  [in, out, string, size_is(ulLength)]
    wchar_t* pszDeviceID,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pszParentDeviceID,
  [in] PNP_RPC_STRING_LEN ulLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszDeviceID: Null-terminated string that contains a device instance ID string that specifies the device instance to create. If the CM_CREATE_DEVNODE_GENERATE_ID flag is set, *pszDeviceID* is assumed to contain simply a device ID. The unique device instance ID string that was generated is returned in this parameter.

If the CM_CREATE_DEVNODE_GENERATE_ID flag is not set, *pszDeviceID* is assumed to contain a device instance ID string.

Once the new device is successfully created, the returned device instance ID string may be used with other methods of this interface to configure or modify the state of the new device.

pszParentDeviceID: device instance ID string of the parent of the new device instance.<35>

ulLength: Size in characters of *pszDeviceID* buffer.

ulFlags: Flags defining the specific action to perform. Possible values:

Value	Meaning
CM_CREATE_DEVNODE_NORMAL 0x00000000	Install later.

Value	Meaning
CM_CREATE_DEVNODE_PHANTOM 0x00000002	Create a device instance, which must not have already been created on the server unless it is an instance of an unregistered firmware mapper device.
CM_CREATE_DEVNODE_GENERATE_ID 0x00000004	Generate a unique device instance ID from the supplied device ID in *pszDeviceID*. If this flag is set, *pszDeviceID* is assumed to contain simply a device ID. The unique device instance ID string that was generated is returned in *pszDeviceID*.
CM_CREATE_DEVNODE_DO_NOT_INSTALL 0x00000008	Create a device instance, but do not install it.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000005 CR_INVALID_DEVINST	The device instance *pszDeviceID* parameter is invalid because the parent device instance, *pszParentDeviceID*, MUST be the root device instance.
0x00000010 CR_ALREADY_SUCH_DEVINST	The device instance *pszDeviceID* already exists.
0x00000013 CR_FAILURE	General failure.
0x00000015 CR_CREATE_BLOCKED	Creation of the device instance is blocked.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available. The *ulLength* parameter value may be too small.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x0000001E CR_INVALID_DEVICE_ID	The specified device ID is invalid.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.26 PNP_DeviceInstanceAction (Opnum 29)

The **PNP_DeviceInstanceAction** method performs operations—for example, setup, enable, or re-enumerate on a device instance. It handles various routines by accepting a major action value and a minor action value.<36>

```
DWORD PNP_DeviceInstanceAction(
  [in] handle_t hBinding,
  [in] unsigned long ulMajorAction,
  [in] unsigned long ulMinorAction,
  [in, string, unique, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pszDeviceInstance1,
  [in, string, unique, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pszDeviceInstance2
);
```

hBinding: A remote procedure call (RPC) binding handle.

ulMajorAction: Specifies the requested action to perform. The following actions are supported. Possible values:

Value	Meaning
PNP_DEVINST_SETUP 0x00000003	Set up the device instance.
PNP_DEVINST_ENABLE 0x00000004	Enable the device instance.
PNP_DEVINST_REENUMERATE 0x00000007	Enumerate the device instance.

ulMinorAction: This value depends on the value of *ulMajorAction* and further defines the specific action to perform.

If the value of *ulMajorAction* is PNP_DEVINST_SETUP, the following values are defined for *ulMinorAction*. Any other value causes a CR_INVALID_FLAG error.

Value	Meaning
CM_SETUP_DEVINST_READY 0x00000000	Instantiates a device and starts it.
CM_SETUP_DEVINST_RESET 0x00000004	Instantiates a device but does not start it.
CM_SETUP_DOWNLOAD 0x00000001	No operation; valid flag, but no action is performed.
CM_SETUP_WRITE_LOG_CONFS 0x00000002	No operation; valid flag, but no action is performed.

If the value of *ulMajorAction* is PNP_DEVINST_REENUMERATE, the following values are defined for *ulMinorAction* and can be combined:

Value	Meaning
CM_REENUMERATE_NORMAL 0x00000000	Specifies default re-enumeration behavior in which re-enumeration occurs synchronously. Equivalent to CM_REENUMERATE_SYNCHRONOUS.
CM_REENUMERATE_SYNCHRONOUS 0x00000001	Re-enumeration should occur synchronously. If this value is used, do not also use CM_REENUMERATE_ASYNCHRONOUS.
CM_REENUMERATE_RETRY_INSTALLATION 0x00000002	The installation should be attempted on any devices that are currently present but not currently installed.
CM_REENUMERATE_ASYNCHRONOUS 0x000000004	Re-enumeration should occur asynchronously. Do not use in combination with CM_REENUMERATE_SYNCHRONOUS.

The *ulMinorAction* parameter is not used for any other values of *ulMajorAction*.

pszDeviceInstance1: A null-terminated string that contains a device instance ID string for the device the operation is performed for.

pszDeviceInstance2: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Not enough memory is available to process this command.
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulMajorAction* parameter is not a recognized value, or the combination of *ulMajorAction* and *ulMinorAction* parameters is invalid.
0x00000005 CR_INVALID_DEVNODE	The *pszDeviceInstance1* parameter is invalid.
0x00000010 CR_ALREADY_SUCH_DEVNODE	The device instance already exists.
0x0000001E CR_INVALID_DEVICE_ID	The specified device ID is invalid.
0x00000025 CR_NO_SUCH_VALUE	The specified value does not exist in the registry.
0x00000028 CR_NOT_DISABLEABLE	The device cannot be disabled.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Return value/code	Description
0x00000034 CR_CALL_NOT_IMPLEMENTED	The call to perform the action specified by *ulMajorAction* is not implemented. The operation cannot be performed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.27 PNP_GetDeviceStatus (Opnum 30)

The **PNP_GetDeviceStatus** method returns the status and problem values for the given device instance.<37>

```
DWORD PNP_GetDeviceStatus(
  [in] handle_t hBinding,
  [in, string, unique, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [out] unsigned long* pulStatus,
  [out] unsigned long* pulProblem,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that contains a device instance ID string for the device for which to return status and problem values.

pulStatus: Returns the device's status. Possible values:

Value	Meaning
DN_ROOT_ENUMERATED 0x00000001	Device instance was enumerated by the root bus enumerator.
DN_DRIVER_LOADED 0x00000002	Device instance has a device driver loaded.
DN_ENUM_LOADED 0x00000004	Device instance has the Register_Enumerator flag, which indicates that an enumeration method is being registered, or needs to be registered.
DN_STARTED 0x00000008	Device instance is currently started.
DN_MANUAL 0x00000010	Device instance was manually installed.
DN_NEED_TO_ENUM 0x00000020	Device instance may need re-enumeration.
DN_DRIVER_BLOCKED 0x00000040	One or more drivers are blocked from loading for this device instance.
DN_NEED_RESTART 0x00000100	System needs to be restarted for this device instance to work properly.

Value	Meaning
DN_CHILD_WITH_INVALID_ID 0x00000200	One or more children have invalid identifier(s).
DN_HAS_PROBLEM 0x00000400	Device instance needs a device installer.
DN_FILTERED 0x00000800	Device instance is filtered.
DN_LEGACY_DRIVER 0x00001000	Device is using a legacy driver.
DN_DISABLEABLE 0x00002000	Device instance can be rebalanced.
DN_REMOVABLE 0x00004000	Device instance can be removed.
DN_PRIVATE_PROBLEM 0x00008000	Device instance has a private problem.
DN_MF_PARENT 0x00010000	Device instance is a multiple-function parent.
DN_MF_CHILD 0x00020000	Device instance is a multiple-function child.
DN_WILL_BE_REMOVED 0x00040000	Device instance will be removed.
DN_NOT_FIRST_TIMEE 0x00080000	Device instance received a configuration enumeration.
DN_STOP_FREE_RES 0x00100000	When child is stopped, free resources.
DN_REBAL_CANDIDATE 0x00200000	Do not skip this device instance during rebalancing.
DN_BAD_PARTIAL 0x00400000	The logical configuration of the device instance does not have the same resources.
DN_NT_ENUMERATOR 0x00800000	Device instance is a Windows NT enumerator.
DN_NT_DRIVER 0x01000000	Device instance is a Windows NT driver.
DN_NEEDS_LOCKING 0x02000000	Device instance needs a lock to resume processing.
DN_ARM_WAKEUP 0x04000000	Device instance can be the wake-up device.
DN_APM_ENUMERATOR 0x08000000	Device instance is an APM-aware enumerator.
DN_APM_DRIVER	Device instance is an APM-aware driver.

Value	Meaning
0x10000000	
DN_SILENT_INSTALL 0x20000000	Device instance was installed silently.
DN_NO_SHOW_IN_DM 0x40000000	Device instance does appear in the device manager.
DN_BOOT_LOG_PROB 0x80000000	A problem occurred during preassignment of the boot log configuration.

pulProblem: Returns the device's problem. Possible values:

Value	Meaning
CM_PROB_NOT_CONFIGURED 0x00000001	No configuration is available for the device.
CM_PROB_OUT_OF_MEMORY 0x00000003	Memory is insufficient to run the device instance.
CM_PROB_INVALID_DATA 0x00000009	Data is invalid.
CM_PROB_FAILED_START 0x0000000A	Device fails while being started.
CM_PROB_NORMAL_CONFLICT 0x0000000C	A configuration conflict has occurred.
CM_PROB_NEED_RESTART 0x0000000E	A restart is required.
CM_PROB_PARTIAL_LOG_CONF 0x00000010	Logical configuration is incomplete.
CM_PROB_UNKNOWN_RESOURCE 0x00000011	Device has an unknown resource type.
CM_PROB_REINSTALL 0x00000012	Reinstallation is required.
CM_PROB_REGISTRY 0x00000013	A registry problem has occurred.
CM_PROB_WILL_BE_REMOVED 0x00000015	Device instance will be removed.
CM_PROB_DISABLED 0x00000016	Device instance is disabled.
CM_PROB_DEVICE_NOT_THERE 0x00000018	Device does not exist.
CM_PROB_NO_VALID_LOG_CONF 0x0000001B	No valid log configuration exists.

Value	Meaning
CM_PROB_FAILED_INSTALL 0x0000001C	Installation failed.
CM_PROB_HARDWARE_DISABLED 0x0000001D	Device is disabled.
CM_PROB_FAILED_ADD 0x0000001F	Driver was not added.
CM_PROB_DISABLED_SERVICE 0x00000020	The service cannot be started either because it is disabled or because it has no enabled devices associated with it.
CM_PROB_TRANSLATION_FAILED 0x00000021	Resource translation failed.
CM_PROB_NO_SOFTCONFIG 0x00000022	Software configuration does not exist.
CM_PROB_BIOS_TABLE 0x00000023	Device is missing in the BIOS table.
CM_PROB_IRQ_TRANSLATION_FAILED 0x00000024	IRQ (interrupt request line) translator failed.
CM_PROB_FAILED_DRIVER_ENTRY 0x00000025	Driver entry method failed.
CM_PROB_DRIVER_FAILED_PRIOR_UNLOAD 0x00000026	Driver should have unloaded, but did not.
CM_PROB_DRIVER_FAILED_LOAD 0x00000027	Driver load was unsuccessful.
CM_PROB_DRIVER_SERVICE_KEY_INVALID 0x00000028	An error occurred while accessing the driver's service key.
CM_PROB_LEGACY_SERVICE_NO_DEVICES 0x00000029	Loaded legacy service did not create any devices.
CM_PROB_DUPLICATE_DEVICE 0x0000002A	Two devices have the same name.
CM_PROB_FAILED_POST_START 0x0000002B	Drivers set the device state to Failed.
CM_PROB_HALTED 0x0000002C	Device failed after being started in user mode.
CM_PROB_PHANTOM 0x0000002D	Device currently exists only in the registry.
CM_PROB_SYSTEM_SHUTDOWN 0x0000002E	System is shutting down.
CM_PROB_HELD_FOR_EJECT 0x0000002F	Device is offline and awaiting removal.

Value	Meaning
CM_PROB_DRIVER_BLOCKED 0x00000030	One or more drivers are blocked from loading.
CM_PROB_REGISTRY_TOO_LARGE 0x00000031	System hive has grown too large.
CM_PROB_SETPROPERTIES_FAILED 0x00000032	Failed to apply one or more registry properties.

A return code of 0x00000000 indicates that there is no problem and that the operation is successful.

ulFlags: Not used.

Note MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	The specified *pulStatus* or *pulProblem* parameter value is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid.
0x00000005 CR_INVALID_DEVNODE	The *pDeviceID* parameter is invalid.
0x00000013 CR_FAILURE	General failure.
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.28 PNP_SetDeviceProblem (Opnum 31)

The **PNP_SetDeviceProblem** method sets the specified problem information for the specified device instance.<38>

```
DWORD PNP_SetDeviceProblem(
  [in] handle_t hBinding,
  [in, string, unique, range(0,PNP_MAX_DEVICE_ID_LEN)]
```

```
    wchar_t* pDeviceID,
  [in] unsigned long ulProblem,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that contains a device instance ID string for the device for which to set a problem.

ulProblem: Device problem to set. For possible values, see section 3.1.4.27.

ulFlags: Flags that specify how the problem should be set. Possible values:

Value	Meaning
CM_SET_DEVNODE_PROBLEM_NORMAL 0x00000000	Set the specified problem only if no other problem currently exists.
CM_SET_DEVNODE_PROBLEM_OVERRIDE 0x00000001	Override the current problem with the new problem.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000005 CR_INVALID_DEVNODE	The *pDeviceID* parameter is invalid.
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.
0x00000013 CR_FAILURE	General failure.
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.29 PNP_DisableDevInst (Opnum 32)

The **PNP_DisableDevInst** method disables the specified device instance.<39>

```
DWORD PNP_DisableDevInst(
```

```
  [in] handle t hBinding,
  [in, string, unique, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in, out, unique] PPNP VETO TYPE pVetoType,
  [in, out, string, unique, max is(ulNameLength), range(0,PNP MAX STRING LEN)]
    wchar t* pszVetoName,
  [in] unsigned long ulNameLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that contains a device instance ID string for the device to disable.

pVetoType: If the removal of the device is vetoed, returns the type of the component that vetoed the operation. For possible values, see section 2.2.11.1.

pszVetoName: If the removal of the device is vetoed, returns a null-terminated string that specifies the name of the component that vetoed the operation.

ulNameLength: Size in characters of the *pszVetoName* buffer.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVINST	The specified device may be invalid, or may be invalid for the operation specified.
0x00000013 CR_FAILURE	General failure.
0x00000017 CR_REMOVE_VETOED	A component such as a service or application refuses to allow removal of this device.
0x00000028 CR_NOT_DISABLEABLE	The device cannot be disabled.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.30 PNP_UninstallDevInst (Opnum 33)

The **PNP_UninstallDevInst** method removes all configuration data and storage locations for the specified device instance.<40>

```
DWORD PNP_UninstallDevInst(
  [in] handle_t hBinding,
  [in, string, unique, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that contains a device instance ID string for the device to uninstall.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The *pDeviceID* parameter is invalid.
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000028 CR_NOT_DISABLEABLE	The device cannot be disabled.
0x00000033 CR_ACCESS_DENIED	Verification of client privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.31 PNP_AddID (Opnum 34)

The **PNP_AddID** method adds a hardware ID or a compatible ID to the registry for the specified device instance.<41>

```
DWORD PNP_AddID(
  [in] handle_t hBinding,
  [in, string, unique, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pszDeviceID,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pszID,
```

```
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszDeviceID: Null-terminated string that contains a device instance ID string for the device for which to add an identifier.

pszID: Hardware ID string or compatible ID string identifier to add.

ulFlags: Specifies the type of identifier to add. Possible values:

Value	Meaning
CM_ADD_ID_HARDWARE 0x00000000	Add the hardware ID.
CM_ADD_ID_COMPATIBLE 0x00000001	Add a compatible ID.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pszDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000013 CR_FAILURE	A general failure occurred, possibly while appending the new identifier to the existing list.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000025 CR_NO_SUCH_VALUE	The specified value does not exist in the registry.
0x00000033	Verification of client access privileges failed.

Return value/code	Description
CR_ACCESS_DENIED	

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.32 PNP_RegisterDriver (Opnum 35)

The **PNP_RegisterDriver** method registers a driver for a device instance and enumerates it.<42>

```
DWORD PNP_RegisterDriver(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pszDeviceID,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszDeviceID: Null-terminated string that contains a device instance ID string for the device for which the driver is being registered. The driver registered for the device is the one that is currently set as the device's CM_DRP_SERVICE well-known property.

ulFlags: Flags for registering the device instance driver. Possible values:

Value	Meaning
CM_REGISTER_DEVICE_DRIVER_STATIC 0x00000000	Static device driver.
CM_REGISTER_DEVICE_DRIVER_DISABLEABLE 0x00000001	Device driver can be disabled. Ignored by the server.
CM_REGISTER_DEVICE_DRIVER_REMOVABLE 0x00000002	Device driver is removable. Ignored by the server.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pszDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules:

Return value/code	Description
	▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.33 PNP_QueryRemove (Opnum 36)

The **PNP_QueryRemove** method queries and removes a device instance. In this context, remove refers to the state of the device, not to the removal of hardware or configuration data.

To process the removal request, the server first queries system components to determine if removal of the device instance is permitted. If the removal is allowed, the system then removes the device instance such that it does not have a status of DN_STARTED. If the removal is not permitted, the method fails with CR_REMOVE_VETOED, and information describing the component that refused to allow the removal of the device instance is returned.<43>

```
DWORD PNP_QueryRemove(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pszDeviceID,
  [in, out, unique] PPNP_VETO_TYPE pVetoType,
  [in, out, string, unique, max_is(ulNameLength), range(0,PNP_MAX_STRING_LEN)]
    wchar_t* pszVetoName,
  [in] unsigned long ulNameLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszDeviceID: Null-terminated string that contains a device instance ID string for the device to query for removal, and then remove the device instance.

pVetoType: If removal of the device is vetoed, returns the type of the component that vetoed the operation. For possible values, see section 2.2.11.1.

pszVetoName: If removal of the device is vetoed, returns a null-terminated string that specifies the name of the component that vetoed the operation.

ulNameLength: Size in characters of the *pszVetoName* buffer.

ulFlags: Flags defining how the query removal should be processed. User interface (UI) dialog boxes are displayed based on if veto type and veto name buffers are supplied. Clients use these flags to determine if a buffer is to be supplied. Possible values:

Value	Meaning
CM_REMOVE_UI_OK 0x00000000	Ignored by the server.
CM_REMOVE_UI_NOT_OK 0x00000001	Ignored by the server.
CM_REMOVE_NO_RESTART 0x00000002	Remove the device instance without restarting the computer.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVINST	The *pszDeviceID* parameter is not valid.
0x00000013 CR_FAILURE	General failure.
0x00000017 CR_REMOVE_VETOED	Some component, such as a service or application, refuses to allow removal of the device. See the *pszVetoName* parameter for the name of the component that issued the veto.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.34 PNP_RequestDeviceEject (Opnum 37)

The **PNP_RequestDeviceEject** method requests that a device be ejected. In this context, eject refers to devices such as disk drives, not to media such as disks and tapes.

To process the eject request, the server first queries system components to determine if removal of the device instance is permitted. If the removal is allowed, the system then removes the device instance such that it does not have a status of DN_STARTED. If the removal is not permitted, the method fails with CR_REMOVE_VETOED, and information describing the component that refused to allow the removal of the device instance is returned.<44>

```
DWORD PNP RequestDeviceEject(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar t* pszDeviceID,
  [in, out, unique] PPNP VETO TYPE pVetoType,
  [in, out, string, unique, max is(ulNameLength), range(0,PNP MAX STRING LEN)]
    wchar t* pszVetoName,
  [in] unsigned long ulNameLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pszDeviceID: Null-terminated string that contains a device instance ID string for the device to eject.

pVetoType: If the removal of the device is vetoed, returns the type of the component that vetoed the operation. For possible values, see section 2.2.11.1.

pszVetoName: If the removal of the device is vetoed, returns a null-terminated string that specifies the name of the component that vetoed the operation.

ulNameLength: Size in characters of the *pszVetoName* buffer.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pszDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000017 CR_REMOVE_VETOED	Some component such as a service or an application refused to allow removal of this device.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.

Return value/code	Description
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed. Undocking, for example, may require a special privilege. If the client is not interactive or is not using the active console session, special load-driver privileges may be required.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.35 PNP_IsDockStationPresent (Opnum 38)

The **PNP_IsDockStationPresent** method determines if a docking station is present.<45>

```
DWORD PNP_IsDockStationPresent(
  [in] handle_t hBinding,
  [in, out, unique] int near* Present
);
```

hBinding: A remote procedure call (RPC) binding handle.

Present: Boolean variable that indicates if a docking station is currently present. Possible values:

Value	Meaning
True 1	docking station is present.
False 0	docking station is not present.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.36 PNP_RequestEjectPC (Opnum 39)

The **PNP_RequestEjectPC** method requests that a computer be ejected (undocked) from its docking station.<46>

```
DWORD PNP_RequestEjectPC(
  [in] handle_t hBinding
);
```

hBinding: A remote procedure call (RPC) binding handle.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.
0x00000013 CR_FAILURE	General failure.
0x00000017 CR_REMOVE_VETOED	Some component, such as a service or application, refuses to allow removal of this device.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.37 PNP_HwProfFlags (Opnum 40)

The **PNP_HwProfFlags** method gets and sets device configuration flags for the specified device instance that are applicable when the system is using the specified hardware profile.<47>

```
DWORD PNP HwProfFlags(
  [in] handle_t hBinding,
  [in] unsigned long ulAction,
  [in, string, ref, range(0,PNP MAX DEVICE ID LEN)]
    wchar t* pDeviceID,
  [in] unsigned long ulConfig,
  [in, out] unsigned long* pulValue,
  [in, out, unique] PPNP_VETO_TYPE pVetoType,
  [in, out, string, unique, max_is(ulNameLength), range(0,PNP_MAX_STRING_LEN)]
    wchar t* pszVetoName,
  [in] unsigned long ulNameLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

ulAction: Action to perform. Possible values:

Value	Meaning
PNP_GET_HWPROFFLAGS 0x00000001	Get the flag value.
PNP_SET_HWPROFFLAGS 0x00000002	Set the flag value.

pDeviceID: Null-terminated string that contains a device instance ID string for the device for which to get or set the hardware profile flag.

ulConfig: Profile for which to get or set the flag. A value of 0 indicates that the current profile should be used.

pulValue: If setting the flag, this value on entry contains the value for the hardware profile flag. If getting the flag, this value returns the current hardware profile flag. Possible values:

Value	Meaning
CSCONFIGFLAG_DISABLED 0x00000001	Disable the device instance in this hardware profile.
CSCONFIGFLAG_DO_NOT_CREATE 0x00000002	Do not allow this device instance to be created in this hardware profile.
CSCONFIGFLAG_DO_NOT_START 0x00000004	Do not allow this device instance to be started in this hardware profile.

pVetoType: Type of the veto. If NULL, no veto information is returned, and the operating system displays the veto information. For possible values, see section 2.2.11.1.

pszVetoName: If the removal of the device is vetoed, returns a null-terminated string that specifies the name of the component that vetoed the operation. If NULL, no veto information is received, and the operating system displays the veto information.

ulNameLength: Size in characters of the *pszVetoName* buffer.

ulFlags: Dependent on the value of *ulAction*. For PNP_GET_HWPROFFLAGS, no flags are valid. For PNP_SET_HWPROFFLAGS, this value can be CM_SET_HW_PROF_FLAGS_UI_NOT_OK.

Value	Meaning
CM_SET_HW_PROF_FLAGS_UI_NOT_OK 0x00000001	Do not display veto user information.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004	The specified *ulFlags* parameter is invalid for this operation.

Return value/code	Description
CR_INVALID_FLAG	
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000022 CR_NEED_RESTART	The system must be restarted for the operation to be completed.
0x00000028 CR_NOT_DISABLEABLE	The device cannot be disabled.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed. Access to set the hardware profile flag value is denied.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.38 PNP_GetHwProfInfo (Opnum 41)

The **PNP_GetHwProfInfo** method returns a structure of information for the specified hardware profile.<48>

```
DWORD PNP_GetHwProfInfo(
  [in] handle_t hBinding,
  [in] unsigned long ulIndex,
  [in, out, ref] HWPROFILEINFO* pHWProfileInfo,
  [in, range(0,sizeof(HWPROFILEINFO))]
    unsigned long ulProfileInfoSize,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

ulIndex: Profile for which to return information.

Note A value of 0xFFFFFFFF indicates that the current profile should be used.

pHWProfileInfo: Pointer to an **HWPROFILEINFO (section 2.2.12.3)** structure that returns profile information.

ulProfileInfoSize: Size of the **HWPROFILEINFO** (section 2.2.12.3) structure.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.
0x00000023 CR_NO_MORE_HW_PROFILES	No more hardware profiles are available.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.39 PNP_AddEmptyLogConf (Opnum 42)

The **PNP_AddEmptyLogConf** method adds an empty logical configuration.<49>

```
DWORD PNP_AddEmptyLogConf(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in] unsigned long ulPriority,
  [out] unsigned long* pulLogConfTag,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that contains a device instance ID string for the device for which to add a logical configuration.

ulPriority: Priority for the new logical configuration. Possible values:

Value	Meaning
LCPRI_FORCECONFIG 0x00000000	Logical configuration is from a forced configuration.

Value	Meaning
LCPRI_BOOTCONFIG 0x00000001	Logical configuration is from a boot configuration.
LCPRI_DESIRED 0x00002000	Logical configuration is the preferred one; it provides better performance.
LCPRI_NORMAL 0x00003000	Logical configuration is the normal one; it provides acceptable performance.
LCPRI_LASTBESTCONFIG 0x00003FFF	For configuration manager (CM) use only; do not use.
LCPRI_SUBOPTIMAL 0x00005000	Logical configuration is less than optimal.
LCPRI_LASTSOFTCONFIG 0x00007FFF	For configuration manager (CM) use only; do not use.
LCPRI_RESTART 0x00008000	Logical configuration needs to restart.
LCPRI_REBOOT 0x00009000	Logical configuration needs to reboot.
LCPRI_POWEROFF 0x0000A000	Indicates that a shutdown or turn-off is required.
LCPRI_HARDRECONFIG 0x0000C000	Indicates that a jumper needs to be changed.
LCPRI_HARDWIRED 0x0000E000	Logical configuration cannot be changed.
LCPRI_IMPOSSIBLE 0x0000F000	Current logical configuration is not possible.
LCPRI_DISABLED 0x0000FFFF	Logical configuration has been disabled.

pulLogConfTag: Returns a tag that identifies what logical configuration this is.

ulFlags: Type of logical configuration to add. Possible values:

Value	Meaning
BASIC_LOG_CONF 0x00000000	Specifies the request list.
FILTERED_LOG_CONF 0x00000001	Specifies the filtered request list.
ALLOC_LOG_CONF 0x00000002	Specifies the Alloc element.
BOOT_LOG_CONF 0x00000003	Specifies the <RM ALLOC> element.

Value	Meaning
FORCED_LOG_CONF 0x00000004	Specifies the FORCED log configuration.
OVERRIDE_LOG_CONF 0x00000005	Specifies the override request list.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Not enough memory is available to process this command.
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed. The caller does not have LoadDriverPrivileges.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.40 PNP_FreeLogConf (Opnum 43)

The **PNP_FreeLogConf** method frees a logical configuration.<50>

```
DWORD PNP_FreeLogConf(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
```

```
    wchar_t* pDeviceID,
  [in] unsigned long ulLogConfType,
  [in] unsigned long ulLogConfTag,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: A null-terminated string that contains a device instance ID string for the device for which to free the logical configuration.

ulLogConfType: The type of logical configuration to free. For possible values, see section 3.1.4.39.

ulLogConfTag: Logical configuration wanted from the specified type of logical configuration.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Not enough memory is available to process this command.
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000007 CR_INVALID_LOG_CONF	The supplied logical configuration parameter is invalid.
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.

Return value/code	Description
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.41 PNP_GetFirstLogConf (Opnum 44)

The **PNP_GetFirstLogConf** method finds the first logical configuration of the specified type for the specified device instance.<51>

```
DWORD PNP_GetFirstLogConf(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in] unsigned long ulLogConfType,
  [out] unsigned long* pulLogConfTag,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that specifies the device instance ID of the device from which to get the logical configuration.

ulLogConfType: Type of logical configuration to get. For possible values, see section 3.1.4.39.

pulLogConfTag: Returns the logical configuration that is wanted from the specified type of logical configuration.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	The value for the *pulLogConfTag* parameter is invalid or missing.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter value is invalid.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components

Return value/code	Description
	▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x0000000E CR_NO_MORE_LOG_CONF	No logical configuration data exists for the specified device.
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A failure occurred when the method tried to read logical configuration information from the registry.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed. Caller does not have LoadDriverPrivileges.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.42 PNP_GetNextLogConf (Opnum 45)

The **PNP_GetNextLogConf** method finds the next logical configuration of the specified type for the specified device instance.<52>

```
DWORD PNP_GetNextLogConf(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in] unsigned long ulLogConfType,
  [in] unsigned long ulCurrentTag,
  [out] unsigned long* pulNextTag,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that specifies the device instance ID of the device from which to get the logical configuration.

ulLogConfType: Type of logical configuration to get. For possible values, see section 3.1.4.39.

ulCurrentTag: Current logical configuration in the enumeration.

pulNextTag: Returns the next logical configuration of the type specified by the *ulLogConfType* parameter for the specified device instance.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000007 CR_INVALID_LOG_CONF	The supplied logical configuration parameter is invalid.
0x0000000E CR_NO_MORE_LOG_CONF	Therefore, neither the device instance nor the logical configuration exists, or the specified logical configuration type contains only resource data or requirements data.
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.43 PNP_GetLogConfPriority (Opnum 46)

The **PNP_GetLogConfPriority** method returns the priority value assigned to the specified logical configuration.<53>

```
DWORD PNP_GetLogConfPriority(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in] unsigned long ulType,
  [in] unsigned long ulTag,
  [out] unsigned long* pPriority,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: A null-terminated device instance ID specifying the device from which to get the priority value of the specified logical configuration.

ulType: Type of logical configuration for which to return priority. For possible values, see section 3.1.4.39.

ulTag: Current logical configuration in the enumeration.

pPriority: Priority for the specified logical configuration. For possible values, see section 3.1.4.39.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000007 CR_INVALID_LOG_CONF	The supplied logical configuration parameter is invalid.
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000025 CR_NO_SUCH_VALUE	The specified value does not exist.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.44 PNP_AddResDes (Opnum 47)

The **PNP_AddResDes** method adds a resource descriptor to the specified logical configuration.<54>

```
DWORD PNP_AddResDes(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in] unsigned long ulLogConfTag,
  [in] unsigned long ulLogConfType,
  [in] RESOURCEID ResourceID,
  [out] unsigned long* pulResourceTag,
  [in, size_is(ResourceLen)] byte far* ResourceData,
  [in] PNP_RPC_BUFFER_SIZE ResourceLen,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: A null-terminated device instance ID specifying the device with the logical configuration to which a resource descriptor is added.

ulLogConfTag: Logical configuration of the specified type for the specified device ID.

ulLogConfType: Type of logical configuration. For possible values, see section 3.1.4.39.

ResourceID: Resource type, a RESOURCEID data type, defined as an unsigned long. Possible values:

Value	Meaning
ResType_All 0x00000000	Return all resource types.
ResType_None 0x00000000	Arbitration always succeeds.
ResType_Mem 0x00000001	Physical address resource.
ResType_IO 0x00000002	Physical I/O address resource.
ResType_DMA 0x00000003	Direct Memory Access (DMA) channels resource.
ResType_IRQ 0x00000004	Interrupt request (IRQ) resource.
ResType_DoNotUse 0x00000005	Spacer used to synchronize subsequent resource types with Windows NT.
ResType_BusNumber 0x00000006	Bus number resource.
ResType_MemLarge	Memory resources >= 4GB.

Value	Meaning
0x00000007	
ResType_Ignored_Bit 0x00008000	Unused.
ResType_ClassSpecific 0x0000FFFF	Class-specific resource.
ResType_Reserved 0x00008000	Reserved for internal use.
ResType_DevicePrivate 0x00008001	Device private data.
ResType_PcCardConfig 0x00008002	PC card configuration data.
ResType_MfCardConfig 0x00008003	MF card configuration data.

pulResourceTag: Returns the index of the resource descriptor within the logical configuration.

ResourceData: Pointer to a structure that specifies resource descriptor data to add to the logical configuration. The format of the structure is determined by the ResourceID type. See section 2.2.12.1.

Returns the resource of the specified type.

ResourceLen: Size in bytes of ResourceData.

ulFlags: Specifies the width of certain variable-size resource descriptor structure fields, as applicable. Possible values:

Value	Meaning
CM_RESDES_WIDTH_32 0x00000001	Client requests a 32-bit resource descriptor structure. Used with ResType_IRQ type resources.
CM_RESDES_WIDTH_64 0x00000002	Client requests a 64-bit resource descriptor structure. Used with ResType_IRQ type resources.

These flags are optional, and this parameter may include no flags (ulFlags = 0). If no flags are specified, the width of the variable-sized resource data supplied is assumed to be 32-bit.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Not enough memory is available to process this command.
0x00000003	A required pointer parameter is invalid.

Return value/code	Description
CR_INVALID_POINTER	
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000006 CR_INVALID_RES_DES	The supplied resource descriptor parameter is invalid.
0x00000007 CR_INVALID_LOG_CONF	The supplied logical configuration parameter is invalid.
0x0000000B CR_INVALID_RESOURCEID	The specified resource id is invalid.
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

Resource descriptors are always added to the end of the list except in the case where a class-specific resource descriptor has already been added. The class-specific resource descriptor always MUST be last. To add new resource descriptors that are not class-specific, place them just before those that are class-specific. A list can include only one class-specific resource descriptor.

3.1.4.45 PNP_FreeResDes (Opnum 48)

The **PNP_FreeResDes** method frees a resource descriptor for the specified logical configuration.<55>

```
DWORD PNP_FreeResDes(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
```

```
    wchar_t* pDeviceID,
  [in] unsigned long ulLogConfTag,
  [in] unsigned long ulLogConfType,
  [in] RESOURCEID ResourceID,
  [in] unsigned long ulResourceTag,
  [out] unsigned long* pulPreviousResType,
  [out] unsigned long* pulPreviousResTag,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: A null-terminated device instance ID specifying the device having the logical configuration with the resource descriptor to be freed.

ulLogConfTag: Logical configuration of the specified type for the specified device ID.

ulLogConfType: Type of logical configuration. For possible values, see section 3.1.4.39.

ResourceID: Resource type. For possible values, see section 3.1.4.44.

ulResourceTag: The tag for the resource type.

pulPreviousResType: Returns the previous resource type. For possible values, see section 3.1.4.44.

pulPreviousResTag: Returns the previous resource of the specified type.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Not enough memory is available to process this command.
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C

Return value/code	Description
	(',')
0x00000006 CR_INVALID_RES_DES	The supplied resource descriptor parameter is invalid.
0x00000007 CR_INVALID_LOG_CONF	The supplied logical configuration parameter is invalid.
0x0000000F CR_NO_MORE_RES_DES	No more resource descriptions are available.
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.46 PNP_GetNextResDes (Opnum 49)

The **PNP_GetNextResDes** method gets the next resource descriptor in the specified logical configuration.<56>

```
DWORD PNP_GetNextResDes(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in] unsigned long ulLogConfTag,
  [in] unsigned long ulLogConfType,
  [in] RESOURCEID ResourceID,
  [in] unsigned long ulResourceTag,
  [out] unsigned long* pulNextResDesTag,
  [out] unsigned long* pulNextResDesType,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: A null-terminated device instance ID specifying the device with the next resource descriptor to be retrieved.

ulLogConfTag: Logical configuration of the specified type for the specified device ID.

ulLogConfType: Type of logical configuration. For possible values, see section 3.1.4.39.

ResourceID: Resource type. For possible values, see section 3.1.4.44.

ulResourceTag: The tag for the resource type.

pulNextResDesTag: Returns the tag for the next resource of the type specified by *pulNextResDesType*.

pulNextResDesType: Returns the next resource type. For possible values, see 3.1.4.44.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Not enough memory is available to process this command.
0x00000003 CR_INVALID_POINTER	A required pointer is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000006 CR_INVALID_RES_DES	The supplied resource descriptor parameter is invalid.
0x00000007 CR_INVALID_LOG_CONF	The supplied logical configuration parameter is invalid.
0x0000000F CR_NO_MORE_RES_DES	No more resource descriptions are available.
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.47 PNP_GetResDesData (Opnum 50)

The **PNP_GetResDesData** method returns the data for the specified resource descriptor.<57>

```
DWORD PNP_GetResDesData(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in] unsigned long ulLogConfTag,
  [in] unsigned long ulLogConfType,
  [in] RESOURCEID ResourceID,
  [in] unsigned long ulResourceTag,
  [out, size_is(BufferLen)] byte far* Buffer,
  [in] PNP_RPC_BUFFER_SIZE BufferLen,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: A null-terminated device instance ID specifying the device with the resource descriptor from which data is retrieved.

ulLogConfTag: Logical configuration of the specified type for the specified device ID.

ulLogConfType: Type of logical configuration. For possible values, see section 3.1.4.39.

ResourceID: Resource type. For possible values, see section 3.1.4.44.

ulResourceTag: The tag for the resource type.

Buffer: Returns resource data of ResourceID type from the logical configuration. See section 2.2.12.1.

BufferLen: Size in bytes of Buffer.

ulFlags: Can specify flags that indicate the width of certain variable-size resource descriptor structure fields. If no such flags are specified, the width of the variable-sized resource data supplied is assumed to be native to the platform of the caller. Possible values:

Value	Meaning
CM_RESDES_WIDTH_32 0x00000001	Client requests a 32-bit IRQ_RESOURCE/IRQ_DES value. Used with ResType_IRQ type resources.
CM_RESDES_WIDTH_64 0x00000002	Client requests a 64-bit IRQ_RESOURCE/IRQ_DES value. Used with ResType_IRQ type resources.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Not enough memory is available to process this command.

Return value/code	Description
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000006 CR_INVALID_RES_DES	The supplied resource descriptor parameter is invalid.
0x00000007 CR_INVALID_LOG_CONF	The supplied logical configuration parameter is invalid.
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.48 PNP_GetResDesDataSize (Opnum 51)

The **PNP_GetResDesDataSize** method returns the data size for the specified resource descriptor.<58>

```
DWORD PNP_GetResDesDataSize(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in] unsigned long ulLogConfTag,
  [in] unsigned long ulLogConfType,
  [in] RESOURCEID ResourceID,
  [in] unsigned long ulResourceTag,
  [out] unsigned long* pulSize,
  [in] unsigned long ulFlags
```

```
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: A null-terminated device instance ID specifying the device with the resource descriptor whose data size is retrieved.

ulLogConfTag: Logical configuration of the specified type for the specified device instance.

ulLogConfType: Type of logical configuration. For possible values, see section 3.1.4.39.

ResourceID: Resource type. For possible values, see section 3.1.4.44.

ulResourceTag: The tag for the resource type.

pulSize: Returns the size in bytes of the buffer required to hold the resource data of ResourceID type from the specified logical configuration, *ulLogConfTag*.

ulFlags: Can specify flags that indicate the width of certain variable-size resource descriptor structure fields. If no such flags are specified, the width of the variable-sized resource data supplied is assumed to be native to the platform of the caller. For possible values, see section 3.1.4.47.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Not enough memory is available to process this command.
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000006 CR_INVALID_RES_DES	The supplied resource descriptor parameter is invalid.
0x00000007	The supplied logical configuration parameter is invalid.

Return value/code	Description
CR_INVALID_LOG_CONF	
0x0000000B CR_INVALID_RESOURCEID	The specified resource id is invalid.
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.49 PNP_ModifyResDes (Opnum 52)

The **PNP_ModifyResDes** method modifies the specified resource descriptor.<59>

```
DWORD PNP_ModifyResDes(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in] unsigned long ulLogConfTag,
  [in] unsigned long ulLogConfType,
  [in] RESOURCEID CurrentResourceID,
  [in] RESOURCEID NewResourceID,
  [in] unsigned long ulResourceTag,
  [in, size_is(ResourceLen)] byte far* ResourceData,
  [in] PNP_RPC_BUFFER_SIZE ResourceLen,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that specifies the device instance ID of the device from which to get the logical configuration.

ulLogConfTag: Logical configuration of the specified type for the specified device instance.

ulLogConfType: Type of the logical configuration. For possible values, see section 3.1.4.39.

CurrentResourceID: Resource type. For possible values, see section 3.1.4.44.

NewResourceID: New resource type. For possible values, see section 3.1.4.44.

ulResourceTag: Resource of the specified type.

ResourceData: Pointer to a structure that specifies new resource descriptor data. The format of the structure is determined by the ResourceID type. See section 2.2.12.1.

ResourceLen: Size of ResourceData in bytes.

ulFlags: Can specify flags that indicate the width of certain variable-size resource descriptor structure fields. If no such flags are specified, the width of the variable-sized resource data supplied is assumed to be native to the platform of the caller. For possible values, see section 3.1.4.47.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Not enough memory is available to process this command.
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000006 CR_INVALID_RES_DES	The supplied resource descriptor parameter is invalid.
0x00000007 CR_INVALID_LOG_CONF	The supplied logical configuration parameter is invalid.
0x0000000B CR_INVALID_RESOURCEID	The specified resource id is invalid.
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.
0x00000013 CR_FAILURE	General failure.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.50 PNP_DetectResourceConflict (Opnum 53)

The **PNP_DetectResourceConflict** method detects conflicts with the specified resource descriptor.<60>

```
DWORD PNP_DetectResourceConflict(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
```

```
    wchar_t* pDeviceID,
  [in] RESOURCEID ResourceID,
  [in, size_is(ResourceLen)] byte far* ResourceData,
  [in] PNP_RPC_BUFFER_SIZE ResourceLen,
  [out] int near* pbConflictDetected,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that contains a device instance ID string specifying the device for which the resource conflict is to be detected.

ResourceID: Resource type. For possible values, see section 3.1.4.44.

ResourceData: Pointer to a structure that specifies resource descriptor data. The format of the structure is determined by the ResourceID type. See section 2.2.12.1.

ResourceLen: Size in bytes of the resource data specified in ResourceData.

pbConflictDetected: Returns a value that indicates if a conflict is detected.

Value	Meaning
TRUE 1	A conflict is detected.
FALSE 0	No conflict is detected.

ulFlags: May specify flags that indicate the width of certain variable-size resource descriptor structure fields. If no such flags are specified, the width of the variable-sized resource data supplied is assumed to be native to the platform of the caller. For possible values, see section 3.1.4.47.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000034 CR_CALL_NOT_IMPLEMENTED	The method is not implemented.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.51 PNP_QueryResConfList (Opnum 54)

The **PNP_QueryResConfList** method returns conflict information for a specified resource.<61>

```
DWORD PNP_QueryResConfList(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
```

```
    wchar_t* pDeviceID,
  [in] RESOURCEID ResourceID,
  [in, size_is(ResourceLen)] byte far* ResourceData,
  [in] PNP_RPC_BUFFER_SIZE ResourceLen,
  [out, size_is(BufferLen)] byte far* Buffer,
  [in] PNP_RPC_BUFFER_SIZE BufferLen,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that contains a device instance ID string for the device for which to return conflict information.

ResourceID: Resource type. For possible values, see section 3.1.4.44.

ResourceData: Pointer to a structure that specifies resource descriptor data. The format of the structure is determined by the ResourceID type. See section 2.2.12.1.

ResourceLen: Size in bytes of ResourceData.

Buffer: Pointer to a **PNP_CONFLICT_LIST** structure that contains the conflict list. See section 2.2.12.1.10.

BufferLen: Size in bytes of *Buffer*. *BufferLen* MUST specify that *Buffer* is at least as large in bytes as: sizeof(PNP_CONFLICT_LIST) + sizeof(PNP_CONFLICT_STRINGS) + (sizeof(wchar_t) * 200). See section 2.2.12.1.10.

ulFlags: Can specify flags that indicate the width of certain variable-size resource descriptor structure fields. If no such flags are specified, the width of the variable-sized resource data supplied is assumed to be native to the platform of the caller. For possible values, see section 3.1.4.47.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c >

Return value/code	Description
	0x7F c == 0x2C (',')
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x0000001F CR_INVALID_DATA	One or more parameters are invalid.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed. Caller does not have LoadDriverPrivileges.
0x0000003B CR_INVALID_STRUCTURE_SIZE	The buffer length specified in *BufferLen* is too small to hold the conflict list data structure.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.52 PNP_GetCustomDevProp (Opnum 61)

The **PNP_GetCustomDevProp** method returns the data for a device instance custom property.<62>

```
DWORD PNP_GetCustomDevProp(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)]
    wchar_t* pDeviceID,
  [in, string, ref, range(0,PNP_MAX_STRING_LEN)]
    wchar_t* CustomPropName,
  [out] unsigned long* pulRegDataType,
  [out, size_is(*pulLength), length_is(*pulTransferLen)]
    byte far* Buffer,
  [out] PNP_RPC_STRING_LEN* pulTransferLen,
  [in, out] PNP_RPC_STRING_LEN* pulLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pDeviceID: Null-terminated string that contains a device instance ID string for the device for which to return the custom property data.

CustomPropName: Name of the property to be returned.<63>

pulRegDataType: Registry data type for the specified property. For possible registry data type values, see section 3.1.4.10.

Buffer: Returns a buffer containing the registry data. If the caller is retrieving the required size, *pulLength* is 0.

pulTransferLen: Length of the data in bytes to copy into *Buffer*. This parameter is used to control how much data is marshaled using RPC between address spaces.

pulLength: Passed in by the caller. On entry, it contains the size in bytes of the buffer. On exit, it contains either the amount of data copied to *Buffer* if a transfer occurred, or the size required to hold the property data if the buffer is too small.

ulFlags: Flag indicating that property values are to be merged. Possible values:

Value	Meaning
CM_CUSTOMDEVPROP_MERGE_MULTISZ 0x00000001	Merge the device-specific REG_SZ or REG_MULTI_SZ property, if present, with the per-hardware identifier REG_SZ or REG_MULTI_SZ property, if present. Thus, the result is always a REG_MULTI_SZ type.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVNODE	The specified *pDeviceID* parameter value is invalid because the device instance string does not conform to one or more of the following device instance path rules: ▪ The total length MUST NOT exceed 200 characters ▪ The path MUST contain exactly three non-empty path components ▪ The path string MUST NOT contain any of the following invalid characters: c <= 0x20 (' ') c > 0x7F c == 0x2C (',')
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available.
0x0000001D CR_REGISTRY_ERROR	A required entry in the registry is missing, or an attempt to write to the registry failed.
0x00000025 CR_NO_SUCH_VALUE	The specified value does not exist.

Return value/code	Description
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.53 PNP_GetVersionInternal (Opnum 62)

The **PNP_GetVersionInternal** method returns the internal version number for the server-side component.<64>

```
DWORD PNP_GetVersionInternal(
  [in] handle_t hBinding,
  [in, out] unsigned short* pwVersion
);
```

hBinding: A remote procedure call (RPC) binding handle.

pwVersion: Returns the internal server version number, with the major version number in the high byte and the minor version number in the low byte.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns the following non-zero error code:

Note CR_SUCCESS has the value 0x00000000.

Return value/code	Description
0x00000013 CR_FAILURE	General failure.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.54 PNP_GetBlockedDriverInfo (Opnum 63)

The **PNP_GetBlockedDriverInfo** method returns the list of drivers that have been blocked from loading on the system since the system was booted.<65>

```
DWORD PNP_GetBlockedDriverInfo(
  [in] handle_t hBinding,
  [out, size_is(*pulLength), length_is(*pulTransferLen)]
    byte far* Buffer,
  [out] PNP_RPC_BUFFER_SIZE* pulTransferLen,
  [in, out] PNP_RPC_BUFFER_SIZE* pulLength,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

Buffer: List of unique string identifiers representing drivers that have been blocked from loading on the system. It is legal to set this to NULL when simply retrieving data size.

pulTransferLen: Length of the data in bytes to copy into *Buffer*. The *pulTransferLen* parameter is used only to control how much data is marshaled between address spaces by means of RPC.

pulLength: Passed in by the caller. On entry, it contains the size in bytes of *Buffer*. On exit, it contains either the amount of data copied to *Buffer* if a transfer occurred, or the size of the buffer required to hold the property data if *Buffer* is too small.

ulFlags: Not used.

Note MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000013 CR_FAILURE	General failure.
0x0000001A CR_BUFFER_SMALL	An output parameter is too small to hold all the data available.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.55 PNP_GetServerSideDeviceInstallFlags (Opnum 64)

The **PNP_GetServerSideDeviceInstallFlags** method returns the server-side device installation flags.<66>

```
DWORD PNP_GetServerSideDeviceInstallFlags(
  [in] handle_t hBinding,
  [out] unsigned long* pulSSDIFlags,
  [in] unsigned long ulFlags
);
```

hBinding: A remote procedure call (RPC) binding handle.

pulSSDIFlags: A pointer supplied by the caller. On successful return, the value is a bitfield that may specify flags defined in the following table. The return value is 0 if no flags have been set.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	R

Value	Description
R SSDI_REBOOT_PENDING	A reboot is required to complete a device configuration operation.

ulFlags: Not used. MUST be set to 0.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *ulFlags* parameter is invalid for this operation.
0x00000013 CR_FAILURE	General failure.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

3.1.4.56 PNP_GetObjectPropKeys (Opnum 65)

The **PNP_GetObjectPropKeys** method is received by the server in an RPC_REQUEST packet. In response, the server returns the properties currently set on the specified object.<67>

```
DWORD PNP_GetObjectPropKeys(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_STRING_LEN)]
    wchar_t* ObjectName,
  [in] unsigned long ObjectType,
  [in, string, unique, range(0,PNP_MAX_CULTURE_NAME_LEN)]
    wchar_t* PropertyCultureName,
  [in, out] PNP_PROP_COUNT* PropertyCount,
  [out] PNP_PROP_COUNT* TransferLen,
  [out, size_is(*PropertyCount), length_is(*TransferLen)]
    DEVPROPKEY* PropertyKeys,
  [in] unsigned long Flags
);
```

hBinding: A remote procedure call (RPC) binding handle.

ObjectName: String identifier describing the object for which the property is to be retrieved. The format of the string depends on the value of *ObjectType*.

ObjectType: Type of object described by the *ObjectName* parameter. Possible values:

Value	Meaning
0x00000002	ObjectName is a device instance ID.
0x00000003	*ObjectName* is a device interface ID.
0x00000004	*ObjectName* is a Device Setup Class GUID.
0x00000005	*ObjectName* is a Device Interface Class GUID.

PropertyCultureName: Culture name for which the specified property key data should be retrieved. The culture name is in the ISO culture name format (for example, en-US). This parameter MUST be NULL when retrieving culture-neutral property keys.

PropertyCount: When receiving data from the client, this parameter is a pointer to the number of **DEVPROPKEY**.

TransferLen: Size in bytes of the *PropertyKeys* parameter.

PropertyKeys: Pointer to an array of **DEVPROPKEY**structures that are being retrieved.

Flags: Flags that modify the behavior of the method. Possible values:

Value	Meaning
0x80000000	Reserved for use by system configuration tasks running on the server.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of .0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Insufficient memory is available to carry out the operation.
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *Flags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVINST	The specified device interface is invalid.
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.
0x00000013 CR_FAILURE	Operation failed to complete. General failure.

Return value/code	Description
0x00000026 CR_WRONG_TYPE	Operation failed to complete. General failure.
0x0000001A CR_BUFFER_SMALL	Buffer is too small.
0x0000001F CR_INVALID_DATA	Data is invalid.
0x00000025 CR_NO_SUCH_VALUE	Value does not exist.
0x00000026 CR_WRONG_TYPE	The specified object type is incorrect.
0x0000002E CR_NO_SUCH_REGISTRY_KEY	Registry key does not exist.
0x0000002F CR_INVALID_MACHINENAME	Supplied computer name is invalid.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.
0x00000035 CR_INVALID_PROPERTY	Property value is invalid.
0x00000037 CR_NO_SUCH_DEVICE_INTERFACE	Device interface does not exist.
0x00000038 CR_INVALID_REFERENCE_STRING	Reference is invalid.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

The values for all In parameters are contained in the **stub_data** field of the RPC_REQUEST packet.

The values for all returned data, including error codes and Out parameters, are contained in the **stub_data** field of the RPC_RESPONSE packet.

3.1.4.57 PNP_GetObjectProp (Opnum 66)

The **PNP_GetObjectProp** method is received by the server in an RPC_REQUEST packet. In response, the server returns the specified object property.<68>

```
DWORD PNP_GetObjectProp(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_STRING_LEN)]
    wchar_t* ObjectName,
  [in] unsigned long ObjectType,
  [in, string, unique, range(0,PNP_MAX_CULTURE_NAME_LEN)]
    wchar_t* PropertyCultureName,
  [in] const DEVPROPKEY* PropertyKey,
  [out] DEVPROPTYPE* PropertyType,
  [in, out] PNP_PROP_SIZE* PropertySize,
```

```
  [out] PNP_PROP_SIZE* TransferLen,
  [out, size_is(*PropertySize), length_is(*TransferLen)]
    byte near* PropertyBuffer,
  [in] unsigned long Flags
);
```

hBinding: A remote procedure call (RPC) binding handle.

ObjectName: String identifier describing the object for which the property is to be retrieved. The format of the string depends on the value of *ObjectType*.

ObjectType: Type of object described by the *ObjectName* parameter. Possible values:

Value	Meaning
0x00000002	*ObjectName* is a device instance ID.
0x00000003	*ObjectName* is a device interface ID.
0x00000004	*ObjectName* is a Device Setup Class GUID.
0x00000005	*ObjectName* is a Device Interface Class GUID.

PropertyCultureName: Culture name for which the specified property data should be retrieved. The culture name is in the ISO culture name format (for example, en-US). This parameter MUST be NULL when retrieving a culture-neutral property.

PropertyKey: Pointer to a **DEVPROPKEY (section 2.2.12.2)** structure that identifies the property to be retrieved.

PropertyType: Pointer to an unsigned long that can specify several different property types, and each property type requires a different format for *PropertyBuffer*. The first 4 bytes of the parameter specify a property type modifier, while the last 4 bytes specify a property type.

These are the possible property type modifier values. These values MUST NOT be specified alone; they MUST be combined with a property type. Note: Property types and property type modifiers MAY be used in the same *PropertyType* parameter; the first 4 bytes (the modifier) modify how the last 4 bytes (the property type) are treated:

Value	Meaning
DEVPROP_TYPEMOD_ARRAY 0x00001000	Array of elements whose data is a fixed size. It MAY be combined with any of the unmodified DEVPROP_TYPE_ types that describe a fixed-size data element. The size of the array is the total size of all data in the array. This value MUST NOT be combined with any other property type modifiers such as DEVPROP_TYPEMOD_LIST.
DEVPROP_TYPEMOD_LIST 0x00002000	List of elements whose data is not a fixed size. It may be combined with any of the unmodified DEVPROP_TYPE types that describe a variable-size data element. The size of the list is dependent on the specific rules for constructing a list of the base element type. This value MUST NOT be combined with any other property type modifiers such as DEVPROP_TYPEMOD_ARRAY.

Possible Property types:

Value	Meaning
DEVPROP_TYPE_EMPTY 0x00000000	Empty property. The property does not exist, and as such, there is no data associated with it. *PropertyBuffer* MUST be set to NULL, and *PropertyBufferSize* MUST be set to 0. This property type value MUST NOT be combined with any property type modifiers.
DEVPROP_TYPE_NULL 0x00000001	NULL property. The property exists, but NULL data is associated with this property. *PropertyBuffer* MUST be set to NULL, and *PropertyBufferSize* MUST be set to 0. This property type value MUST NOT be combined with any property type modifiers.
DEVPROP_TYPE_SBYTE 0x00000002	Signed byte. *PropertyBuffer* specifies a pointer to a 1-byte signed integer. *PropertyBufferSize* MUST be set to 1.
DEVPROP_TYPE_BYTE 0x00000003	Unsigned byte. *PropertyBuffer* specifies a 1-byte unsigned integer. *PropertyBufferSize* MUST be set to 1.
DEVPROP_TYPE_INT16 0x00000004	Signed integer. *PropertyBuffer* specifies a 16-bit signed integer. *PropertyBufferSize* MUST be set to 2.
DEVPROP_TYPE_UINT16 0x00000005	Unsigned integer. *PropertyBuffer* specifies a 16-bit unsigned integer. *PropertyBufferSize* MUST be set to 2.
DEVPROP_TYPE_INT32 0x00000006	Signed integer. *PropertyBuffer* specifies a 32-bit unsigned integer. *PropertyBufferSize* MUST be set to 4.
DEVPROP_TYPE_UINT32 0x00000007	Unsigned integer. *PropertyBuffer* specifies a 32-bit unsigned integer. *PropertyBufferSize* MUST be set to 4.
DEVPROP_TYPE_INT64 0x00000008	Signed integer. *PropertyBuffer* specifies a 64-bit unsigned integer. *PropertyBufferSize* MUST be set to 8.
DEVPROP_TYPE_UINT64 0x00000009	Unsigned integer. *PropertyBuffer* specifies a 64-bit unsigned integer. *PropertyBufferSize* MUST be set to 8.
DEVPROP_TYPE_FLOAT 0x0000000A	Floating point value. *PropertyBuffer* specifies a 32-bit IEEE floating-point value. *PropertyBufferSize* MUST be set to 4.
DEVPROP_TYPE_DOUBLE 0x0000000B	Floating point value. *PropertyBuffer* specifies a 64-bit IEEE floating-point value. *PropertyBufferSize* MUST be set to 8.
DEVPROP_TYPE_DECIMAL	Decimal structure. *PropertyBuffer* specifies a DECIMAL structure. *PropertyBufferSize* MUST

Value	Meaning
0x0000000C	be set to 16.
DEVPROP_TYPE_GUID 0x0000000D	A globally unique identifier (GUID). *PropertyBuffer* specifies a GUID. *PropertyBufferSize* MUST be set to 16.
DEVPROP_TYPE_CURRENCY 0x0000000E	Currency value. *PropertyBuffer* specifies an 8-byte two's complement integer (scaled by 10,000). This type is commonly used for currency amounts. *PropertyBufferSize* MUST be set to 8.
DEVPROP_TYPE_DATE 0x0000000F	Date value. *PropertyBuffer* specifies a 64-bit floating-point number representing the number of days (not seconds) since December 31, 1899. For example, January 1, 1900, is 2.0, January 2, 1900, is 3.0, and so on. *PropertyBufferSize* MUST be set to 8.
DEVPROP_TYPE_FILETIME 0x00000010	Time value. *PropertyBuffer* specifies a FILETIME structure. *PropertyBufferSize* MUST be set to 8.
DEVPROP_TYPE_BOOLEAN 0x00000011	Boolean value. *PropertyBuffer* specifies a Boolean value represented by an 8-bit signed integer containing 0 (FALSE) or -1 (TRUE). *PropertyBufferSize* MUST be set to 1.
DEVPROP_TYPE_STRING 0x00000012	String value. *PropertyBuffer* specifies a null-terminated Unicode string. *PropertyBufferSize* specifies the size of the string, including the terminating null character. Because the data has a variable size, this property type MUST NOT be combined with the DEVPROP_TYPEMOD_ARRAY property type modifier.
DEVPROP_TYPE_SECURITY_DESCRIPTOR 0x00000013	Security descriptor value. *PropertyBuffer* specifies a self-relative binary security descriptor (as specified in [MS-DTYP]. *PropertyBufferSize* specifies the size in bytes of the data. Because the data has a variable size, this property type MUST NOT be combined with the DEVPROP_TYPEMOD_ARRAY property type modifier.
DEVPROP_TYPE_SECURITY_DESCRIPTOR_STRING 0x00000014	Security descriptor string. *PropertyBuffer* specifies a null-terminated Unicode string that describes a security descriptor using the Security Descriptor Definition Language format (as specified in [MS-DTYP]. *PropertyBufferSize* specifies the size of the data in bytes, including the size of the terminating null character. Because the data has a variable size, this property type MUST NOT be combined with the

Value	Meaning
	DEVPROP_TYPEMOD_ARRAY property type modifier.
DEVPROP_TYPE_DEVPROPKEY 0x00000015	A **DEVPROPKEY** (section 2.2.12.2) structure. *PropertyBuffer* specifies a **DEVPROPKEY** (section 2.2.12.2) structure. *PropertyBufferSize* MUST be set to 18.
DEVPROP_TYPE_DEVPROPTYPE 0x00000016	A **DEVPROPKEY** (section 2.2.12.2) structure. *PropertyBuffer* specifies a DEVPROPTYPE value, which in Windows is an unsigned long. The possible values of a DEVPROPTYPE are those specified in this table. *PropertyBufferSize* MUST be set to 2.
DEVPROP_TYPE_ERROR 0x00000017	Error code. *PropertyBuffer* specifies a 32-bit Win32 system error code value, as specified in [MS-ERREF]. *PropertyBufferSize* MUST be set to 4.
DEVPROP_TYPE_NTSTATUS 0x00000018	Error code. *PropertyBuffer* specifies a 32-bit NTSTATUS error code value, as specified in [MS-ERREF]. *PropertyBufferSize* MUST be set to 4.
DEVPROP_TYPE_STRING_INDIRECT 0x00000019	Indirect string reference. *PropertyBuffer* specifies a null-terminated Unicode string that describes an indirect string reference. An indirect string reference describes a resource from which the actual string data can be retrieved. The indirect string reference MUST be in one of two formats. @[path\]filename,-resourceID The string is extracted from the named module, using the value of resourceId (excluding the required minus sign) as a resource identifier. The string resource is loaded from the module resource section that best matches one of the caller's configured preferred UI languages. The path is optional. If no path is specified, the module must reside in a directory in the system-defined search path. @InfName,%strkey% The string is extracted from the [Strings] section of the named INF in the %windir%\inf directory. The strkey token identifier is expected to match the key of some line in the INF Strings section that best matches one of the caller's preferred UI languages. If no language-specific [Strings] sections exist, the default [Strings] section is used. Property data set using this type MUST be in one of the indirect string formats described above. When a property of this type is retrieved by a caller, an attempt MUST be made to locate the string specified by the

Value	Meaning
	property data. If the string can be retrieved, it MUST be returned to the caller, and the type of data MUST be returned as DEVPROP_TYPE_STRING. If the string cannot be located, the indirect string property data MUST be returned to the caller, and the type MUST remain DEVPROP_TYPE_STRING_INDIRECT. This property type MUST NOT be combined with any property type modifiers.

PropertySize: When receiving data from the client, this parameter is a pointer to the size in bytes of the *PropertyBuffer* parameter. When receiving data from the server, this parameter is either a pointer to the actual size of the data in the *PropertyBuffer* parameter (if successful) or the size required (if not successful because the buffer is too small).

TransferLen: Size in bytes of the *PropertyBuffer*.

PropertyBuffer: Pointer to a buffer containing data for the property being retrieved. The format of the buffer is determined by the property type. For the property types and the expected format of the buffer, see *PropertyType* above.

Flags: Flags that modify the behavior of the method. Possible values:

Value	Meaning
0x80000000	Reserved for use by system configuration tasks running on the server.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Insufficient memory is available to carry out the operation.
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *Flags* parameter is invalid for this operation.
0x00000005 CR_INVALID_DEVINST	The specified device interface is invalid.
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.
0x00000013 CR_FAILURE	Operation failed to complete. General failure.
0x00000026 CR_WRONG_TYPE	The specified object type is incorrect.

Return value/code	Description
0x0000001A CR_BUFFER_SMALL	Buffer is too small.
0x0000001F CR_INVALID_DATA	Data is invalid.
0x00000025 CR_NO_SUCH_VALUE	Value does not exist.
0x0000002E CR_NO_SUCH_REGISTRY_KEY	Registry key does not exist.
0x0000002F CR_INVALID_MACHINENAME	Supplied computer name is invalid.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.
0x00000035 CR_INVALID_PROPERTY	Property value is invalid.
0x00000037 CR_NO_SUCH_DEVICE_INTERFACE	Device Interface does not exist.
0x00000038 CR_INVALID_REFERENCE_STRING	Reference is invalid.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

The values for all In parameters are contained in the **stub_data** field of the RPC_REQUEST packet.

The values for all returned data, including error codes and Out parameters, are contained in the **stub_data** field of the RPC_RESPONSE packet.

3.1.4.58 PNP_SetObjectProp (Opnum 67)

The **PNP_SetObjectProp** method is received by the server in an RPC_REQUEST packet. In response, the server sets the specified object property.<69>

```
DWORD PNP_SetObjectProp(
  [in] handle_t hBinding,
  [in, string, ref, range(0,PNP_MAX_STRING_LEN)]
    wchar_t* ObjectName,
  [in] unsigned long ObjectType,
  [in, string, unique, range(0,PNP_MAX_CULTURE_NAME_LEN)]
    wchar_t* PropertyCultureName,
  [in] const DEVPROPKEY* PropertyKey,
  [in] DEVPROPTYPE PropertyType,
  [in] PNP_PROP_SIZE PropertySize,
  [in, unique, size_is(PropertySize)]
    byte near* PropertyBuffer,
  [in] unsigned long Flags
);
```

hBinding: A remote procedure call (RPC) binding handle.

ObjectName: String identifier describing the object for which the property is to be set. The format of the string depends on the value of *ObjectType*.

ObjectType: Type of object described by the *ObjectName* parameter. Possible values:

Value	Meaning
0x00000002	*ObjectName* is a device instance ID.
0x00000003	*ObjectName* is a device interface ID.
0x00000004	*ObjectName* is a Device Setup Class GUID.
0x00000005	*ObjectName* is a Device Interface Class GUID.

PropertyCultureName: Culture name for which the specified property data should be set. The culture name is in the ISO culture name format (for example, en-US). This parameter MUST be NULL when storing a culture-neutral property.

PropertyKey: Pointer to a **DEVPROPKEY (section 2.2.12.2)** structure that identifies the property to be set.

PropertyType: *PropertyType* pointer to an unsigned long that can specify several different property types, and each property type requires a different format for *PropertyBuffer*. For a full description of this member and its possible values, see the *PropertyType* description in section 3.1.4.57.

PropertySize: Size in bytes of the *PropertyBuffer*.

PropertyBuffer: Pointer to a buffer containing data for the property being set. The format of the buffer is determined by the property type. For a description of the property types and the expected format of the buffer, see the *PropertyType* description in section 3.1.4.57.

Flags: Flags that modify the behavior of the method. Possible values:

Value	Meaning
0x80000000	Reserved for use by system configuration tasks running on the server.

Return Values: The method returns CR_SUCCESS on success; otherwise, it returns one of the following non-zero error codes:

Note CR_SUCCESS has a value of 0x00000000.

Return value/code	Description
0x00000002 CR_OUT_OF_MEMORY	Insufficient memory is available to carry out the operation.
0x00000003 CR_INVALID_POINTER	A required pointer parameter is invalid.
0x00000004 CR_INVALID_FLAG	The specified *Flags* parameter is invalid for this operation.

Return value/code	Description
0x00000005 CR_INVALID_DEVINST	The specified device interface is invalid.
0x0000000D CR_NO_SUCH_DEVINST	The specified device instance does not correspond to a present device.
0x00000013 CR_FAILURE	Operation failed to complete. General failure.
0x0000001A CR_BUFFER_SMALL	Buffer is too small.
0x0000001F CR_INVALID_DATA	Data is invalid.
0x00000025 CR_NO_SUCH_VALUE	Value does not exist.
0x00000026 CR_WRONG_TYPE	The specified object type is incorrect.
0x0000002E CR_NO_SUCH_REGISTRY_KEY	Registry key does not exist.
0x0000002F CR_INVALID_MACHINENAME	Supplied computer name is invalid.
0x00000033 CR_ACCESS_DENIED	Verification of client access privileges failed.
0x00000035 CR_INVALID_PROPERTY	Property value is invalid.
0x00000037 CR_NO_SUCH_DEVICE_INTERFACE	Device Interface does not exist.
0x00000038 CR_INVALID_REFERENCE_STRING	Reference is invalid.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol. See [MS-RPCE].

The values for all In parameters are contained in the **stub_data** field of the RPC_REQUEST packet.

The values for all returned data, including error codes and Out parameters, are contained in the **stub_data** field of the RPC_RESPONSE packet.

3.1.5 Timer Events

This protocol has no timer events.

3.1.6 Other Local Events

This protocol has no other local events.

3.2 Client Details

The client establishes a connection to the server and sends messages to manage devices and other abstract objects on the remote machine.

3.2.1 Abstract Data Model

No abstract data model is used.

3.2.2 Timers

This protocol has no timers.

3.2.3 Initialization

The client initializes by creating a remote procedure call (RPC) binding handle to the PNPR Protocol interface on the remote computer. For how to get a client-side RPC binding handle, see [MS-DCOM] section 3.1.4.

3.2.4 Message Processing Events and Sequencing Rules

This protocol MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 6.0, as specified in [MS-RPCE] section 3.<70>

3.2.5 Timer Events

This protocol has no timer events.

3.2.6 Other Local Events

This protocol has no other local events.

4 Protocol Examples

For most methods, this is a simple request-response protocol. For every method that the server receives, it executes the method and returns a completion. The client simply returns the completion status to the caller.

The following sections provide examples of how a Plug and Play Remote (PNPR) Protocol client and server communicate in common scenarios.

4.1 Retrieving a List of Devices

The following example depicts a client retrieving a list of devices on the server:

1. The client requests the size required to hold a multi-sz list of device instance ID strings, representing the devices on the server by calling the **PNP_GetDeviceListSize (section 3.1.4.8)** method, passing in a reference to an UNSIGNED LONG parameter in which to store the required size.

2. The server processes the request by estimating the largest possible size in characters required to hold the multi-sz list of device instance ID strings. The server returns CR_SUCCESS (0x00000000) to acknowledge that the operation is successful.

3. The client requests the list of device instance ID strings by calling the **PNP_GetDeviceList (section 3.1.4.7)** method, passing in a reference to a buffer that is at least as large as the size returned from the previous call to **PNP_GetDeviceListSize**.

4. The server processes the request by supplying the multi-sz list of devices, and returns CR_SUCCESS (0x00000000) to acknowledge that the operation is successful.

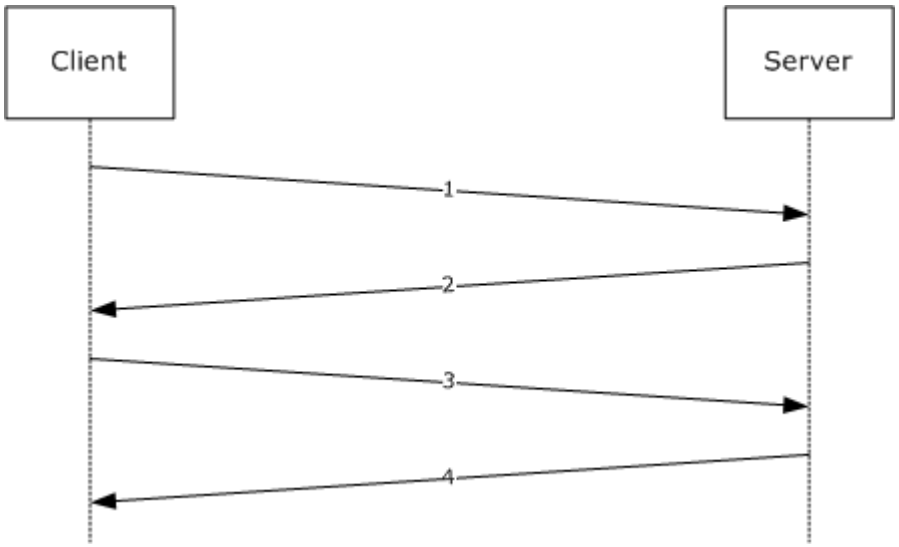

Figure 1: Message exchange to retrieve list of devices

4.2 Retrieving Status and Problem Values of a Device

The following example depicts a client retrieving the status and problem values for a device on the server:

1. The client calls the **PNP_GetDeviceStatus (section 3.1.4.27)** method, passing in references to UNSIGNED LONG parameters in which to store the status and problem values.

2. The server returns the problem and status attributes of the device, and returns CR_SUCCESS (0x00000000) to acknowledge that the operation is successful.

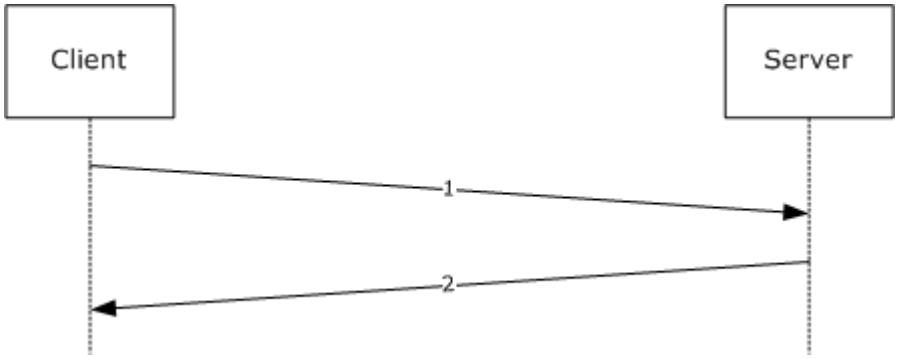

Figure 2: Message exchange to retrieve status and problem values of a device

5 Security

The following sections specify security considerations for administrators.

5.1 Security Considerations for Implementers

This protocol allows any user to connect to the server. Therefore, any security bug in the server implementation can be exploitable. The server implementation SHOULD enforce security on each method.

6 Appendix A: Full IDL

For ease of implementation, the full **Interface Definition Language (IDL)** is provided below where ms-dtyp.idl is the IDL specified in [MS-DTYP]Appendix A.

```
import "ms-dtyp.idl";

 [
    uuid(8D9F4E40-A03D-11CE-8F69-08003E30051B),
    version(1.0),
#ifdef __midl
    ms_union,
#endif // __midl
    pointer_default(unique)
]

interface pnp

{

const unsigned long PNP_MAX_STRING_LEN        = 32767;
const unsigned long PNP_MAX_DEVICE_ID_LEN     = 200;
const unsigned long PNP_MAX_GUID_STRING_LEN   = 39;
const unsigned long PNP_MAX_DEVINTERFACE_LEN  = PNP_MAX_STRING_LEN;
const unsigned long PNP_MAX_CULTURE_NAME_LEN  = 85;
const unsigned long PNP_MAX_CM_PATH           = 360;
const unsigned long PNP_MAX_PROP_SIZE         = 65534;
const unsigned long PNP_MAX_PROP_COUNT        = 32767;
const unsigned long PNP_MAX_BUFFER_SIZE       = 0xF42400;


typedef [range(0,PNP_MAX_PROP_SIZE)] unsigned long PNP_PROP_SIZE;
typedef PNP_PROP_SIZE *PPNP_PROP_SIZE;
typedef [range(0,PNP_MAX_PROP_COUNT)] unsigned long PNP_PROP_COUNT;
typedef PNP_PROP_COUNT *PPNP_PROP_COUNT;
typedef [range(0,PNP_MAX_STRING_LEN)] unsigned long
     PNP_RPC_STRING_LEN;
typedef PNP_RPC_STRING_LEN *PPNP_RPC_STRING_LEN;
typedef [range(0,PNP_MAX_BUFFER_SIZE)] unsigned long
     PNP_RPC_BUFFER_SIZE;
typedef PNP_RPC_BUFFER_SIZE *PPNP_RPC_BUFFER_SIZE;

typedef unsigned long RESOURCEID;
typedef unsigned long DEVPROPTYPE;

// Enums

typedef enum _PPNP_VETO_TYPE
{
  PNP_VetoTypeUnknown = 0,
  PNP_VetoLegacyDevice = 1,
  PNP_VetoPendingClose = 2,
  PNP_VetoWindowsApp = 3,
  PNP_VetoWindowsService = 4,
  PNP_VetoOutstandingOpen = 5,
  PNP_VetoDevice = 6,
  PNP_VetoDriver = 7,
```

```
  PNP_VetoIllegalDeviceRequest = 8,
  PNP_VetoInsufficientPower = 9,
  PNP_VetoNonDisableable = 10,
  PNP_VetoLegacyDriver = 11,
  PNP_VetoInsufficientRights = 12
} *PPNP_VETO_TYPE;


// Structures

typedef struct _DEVPROPKEY {
  GUID fmtid;
  unsigned long pid;
} DEVPROPKEY;

typedef struct {
  unsigned long HWPI_ulHWProfile;
  wchar_t   HWPI_szFriendlyName[80];
  DWORD   HWPI_dwFlags;
} HWPROFILEINFO;




// Methods

void Opnum00NotUsedOnWire(void);

void Opnum01NotUsedOnWire(void);

DWORD PNP_GetVersion(
    [in]         handle_t                hBinding,
    [out]        unsigned short                   *pVersion
    );

DWORD PNP_GetGlobalState(
    [in]         handle_t                hBinding,
    [out]        unsigned long *                   pulState,
    [in]        unsigned long                    ulFlags
    );

void PNP_LocalOnlyOpnum04(void);

void PNP_LocalOnlyOpnum05(void);

DWORD PNP_ValidateDeviceInstance(
    [in]         handle_t                hBinding,
    [in, string, ref, range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
         pDeviceID,
    [in]         unsigned long                    ulFlags
    );

DWORD PNP_GetRootDeviceInstance(
    [in]         handle_t                hBinding,
    [out,string,size_is(ulLength)] wchar_t * pDeviceID,
    [in]         PNP_RPC_STRING_LEN       ulLength
    );
```

```
DWORD PNP_GetRelatedDeviceInstance(
    [in]         handle_t                hBinding,
    [in]         unsigned long                   ulRelationship,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
         pDeviceID,
    [out,string,size_is(*pulLength)] wchar_t * pRelatedDeviceID,
    [in,out]     PNP_RPC_STRING_LEN      *pulLength,
    [in]         unsigned long                   ulFlags
    );

DWORD PNP_EnumerateSubKeys(
    [in]         handle_t                hBinding,
    [in]         unsigned long                   ulBranch,
    [in]         unsigned long                   ulIndex,
    [out,string,size_is(ulLength)] wchar_t *  Buffer,
    [in]         PNP_RPC_STRING_LEN       ulLength,
    [out]        PNP_RPC_STRING_LEN      *pulRequiredLen,
    [in]         unsigned long                   ulFlags
    );

DWORD PNP_GetDeviceList(
    [in]         handle_t                hBinding,
    [in,string,unique,range(0,PNP_MAX_STRING_LEN)] wchar_t *
        pszFilter,
    [out,size_is(*pulLength),length_is(*pulLength)] wchar_t *
        Buffer,
    [in,out]     PNP_RPC_BUFFER_SIZE     *pulLength,
    [in]         unsigned long                   ulFlags
    );

DWORD PNP_GetDeviceListSize(
    [in]         handle_t                hBinding,
    [in,string,unique,range(0,PNP_MAX_STRING_LEN)] wchar_t *
        pszFilter,
    [out]        PNP_RPC_BUFFER_SIZE     *pulLen,
    [in]         unsigned long                   ulFlags
    );

DWORD PNP_GetDepth(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pszDeviceID,
    [out]        unsigned long *                  pulDepth,
    [in]        unsigned long                  ulFlags
    );

DWORD PNP_GetDeviceRegProp(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long                   ulProperty,
    [in,out]     unsigned long *                  pulRegDataType,
    [out,size_is(*pulTransferLen),length_is(*pulTransferLen)] byte
        far * Buffer,
    [in,out]     PNP_PROP_SIZE           *pulTransferLen,
    [in,out]     PNP_PROP_SIZE           *pulLength,
    [in]        unsigned long                   ulFlags
```

```
    );

DWORD PNP_SetDeviceRegProp(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long                   ulProperty,
    [in]        unsigned long                   ulDataType,
    [in,size_is(ulLength)] byte far *         Buffer,
    [in]         PNP_PROP_SIZE           ulLength,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_GetClassInstance(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [out,string,size_is(ulLength)] wchar_t * pszClassInstance,
    [in]         PNP_RPC_STRING_LEN       ulLength
    );

DWORD PNP_CreateKey(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_CM_PATH)] wchar_t * pszSubKey,
    [in]         DWORD                   samDesired,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_DeleteRegistryKey(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pszDeviceID,
    [in,string,ref,range(0,PNP_MAX_CM_PATH)] wchar_t * pszParentKey,
    [in,string,ref,range(0,PNP_MAX_CM_PATH)] wchar_t * pszChildKey,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_GetClassCount(
    [in]         handle_t                hBinding,
    [out]        unsigned long *                  pulClassCount,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_GetClassName(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_GUID_STRING_LEN)] wchar_t *
        pszClassGuid,
    [out,string,size_is(*pulLength)] wchar_t * Buffer,
    [in,out]     PNP_RPC_STRING_LEN      *pulLength,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_DeleteClassKey(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_GUID_STRING_LEN)] wchar_t *
        pszClassGuid,
    [in]        unsigned long                   ulFlags
    );
```

```
DWORD PNP_GetInterfaceDeviceAlias(
    [in]         handle_t                 hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVINTERFACE_LEN)] wchar_t *
        pszInterfaceDevice,
    [in]         GUID *                   AliasInterfaceGuid,
    [out,string,size_is(*pulTransferLen)] wchar_t *
        pszAliasInterfaceDevice,
    [in,out]     PNP_RPC_STRING_LEN      *pulLength,
    [in,out]     PNP_RPC_STRING_LEN      *pulTransferLen,
    [in]        unsigned long                    ulFlags
    );

DWORD PNP_GetInterfaceDeviceList(
    [in]         handle_t                 hBinding,
    [in]         GUID *                   InterfaceGuid,
    [in,string,unique,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pszDeviceID,
    [out,size_is(*pulLength),length_is(*pulLength)]  wchar_t *
        Buffer,
    [in,out]     PNP_RPC_BUFFER_SIZE     *pulLength,
    [in]        unsigned long                    ulFlags
    );

DWORD PNP_GetInterfaceDeviceListSize(
    [in]         handle_t                 hBinding,
    [out]        PNP_RPC_BUFFER_SIZE     *pulLen,
    [in]         GUID *                   InterfaceGuid,
    [in,string,unique,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pszDeviceID,
    [in]        unsigned long                    ulFlags
    );

DWORD PNP_RegisterDeviceClassAssociation(
    [in]         handle_t                 hBinding,
    [in,string,unique,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pszDeviceID,
    [in]         GUID *                   InterfaceGuid,
    [in,string,unique,range(0,PNP_MAX_STRING_LEN)] wchar_t *
        pszReference,
    [out,string,size_is(*pulTransferLen)] wchar_t * pszSymLink,
    [in,out]     PNP_RPC_STRING_LEN      *pulLength,
    [in,out]     PNP_RPC_STRING_LEN      *pulTransferLen,
    [in]        unsigned long                    ulFlags
    );

DWORD PNP_UnregisterDeviceClassAssociation(
    [in]         handle_t                 hBinding,
    [in,string,unique,range(0,PNP_MAX_DEVINTERFACE_LEN)] wchar_t *
        pszInterfaceDevice,
    [in]        unsigned long                    ulFlags
    );

DWORD PNP_GetClassRegProp(
    [in]         handle_t                 hBinding,
    [in,string,ref,range(0,PNP_MAX_GUID_STRING_LEN)] wchar_t *
        pszClassGuid,
    [in]        unsigned long                    ulProperty,
```

```
    [in,out]     unsigned long *                  pulRegDataType,
    [out,size_is(*pulTransferLen),length_is(*pulTransferLen)] byte
        far * Buffer,
    [in,out]     PNP_PROP_SIZE           *pulTransferLen,
    [in,out]     PNP_PROP_SIZE           *pulLength,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_SetClassRegProp(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_GUID_STRING_LEN)] wchar_t *
        pszClassGuid,
    [in]        unsigned long                   ulProperty,
    [in]        unsigned long                   ulDataType,
    [in,size_is(ulLength)] byte far *         Buffer,
    [in]         PNP_PROP_SIZE           ulLength,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_CreateDevInst(
    [in]         handle_t                hBinding,
    [in,out,string,size_is(ulLength)] wchar_t * pszDeviceID,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pszParentDeviceID,
    [in]         PNP_RPC_STRING_LEN      ulLength,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_DeviceInstanceAction(
    [in]         handle_t                hBinding,
    [in]        unsigned long                   ulMajorAction,
    [in]        unsigned long                   ulMinorAction,
    [in,string,unique,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pszDeviceInstance1,
    [in,string,unique,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pszDeviceInstance2
    );

DWORD PNP_GetDeviceStatus(
    [in]         handle_t                hBinding,
    [in,string,unique,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [out]        unsigned long *                   pulStatus,
    [out]        unsigned long *                   pulProblem,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_SetDeviceProblem(
    [in]         handle_t                hBinding,
    [in,string,unique,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long                   ulProblem,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_DisableDevInst(
    [in]         handle_t                hBinding,
    [in,string,unique,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
```

```
        pDeviceID,
    [in,out,unique] PPNP_VETO_TYPE        pVetoType,
    [in,out,string,unique,max_is(ulNameLength),
        range(0,PNP_MAX_STRING_LEN)] wchar_t * pszVetoName,
    [in]        unsigned long                  ulNameLength,
    [in]        unsigned long                  ulFlags
    );

DWORD PNP_UninstallDevInst(
    [in]         handle_t                hBinding,
    [in,string,unique,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long                 ulFlags
    );

DWORD PNP_AddID(
    [in]         handle_t                hBinding,
    [in,string,unique,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pszDeviceID,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t * pszID,
    [in]        unsigned long                  ulFlags
    );

DWORD PNP_RegisterDriver(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pszDeviceID,
    [in]        unsigned long                 ulFlags
    );

DWORD PNP_QueryRemove(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pszDeviceID,
    [in,out,unique] PPNP_VETO_TYPE        pVetoType,
    [in,out,string,unique,max_is(ulNameLength),
        range(0,PNP_MAX_STRING_LEN)] wchar_t * pszVetoName,
    [in]        unsigned long                  ulNameLength,
    [in]        unsigned long                  ulFlags
    );

DWORD PNP_RequestDeviceEject(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pszDeviceID,
    [in,out,unique] PPNP_VETO_TYPE        pVetoType,
    [in,out,string,unique,max_is(ulNameLength),
        range(0,PNP_MAX_STRING_LEN)] wchar_t * pszVetoName,
    [in]        unsigned long                  ulNameLength,
    [in]        unsigned long                  ulFlags
    );

DWORD PNP_IsDockStationPresent(
    [in]         handle_t                hBinding,
    [in,out,unique] int near *                Present
    );

DWORD PNP_RequestEjectPC(
```

```
    [in]         handle_t                hBinding
    );

DWORD PNP_HwProfFlags(
    [in]         handle_t                hBinding,
    [in]        unsigned long                   ulAction,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long                   ulConfig,
    [in,out]     unsigned long *                  pulValue,
    [in,out,unique] PPNP_VETO_TYPE        pVetoType,
    [in,out,string,unique,max_is(ulNameLength),
        range(0,PNP_MAX_STRING_LEN)] wchar_t * pszVetoName,
    [in]        unsigned long                   ulNameLength,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_GetHwProfInfo(
    [in]         handle_t                hBinding,
    [in]        unsigned long                   ulIndex,
    [in,out,ref] HWPROFILEINFO           *pHWProfileInfo,
    [in,range(0,sizeof(HWPROFILEINFO))]unsigned long
        ulProfileInfoSize,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_AddEmptyLogConf(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long                   ulPriority,
    [out]        unsigned long *                  pulLogConfTag,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_FreeLogConf(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long                   ulLogConfType,
    [in]        unsigned long                   ulLogConfTag,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_GetFirstLogConf(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long                   ulLogConfType,
    [out]        unsigned long *                  pulLogConfTag,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_GetNextLogConf(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long                   ulLogConfType,
```

```
    [in]        unsigned long                   ulCurrentTag,
    [out]        unsigned long *                  pulNextTag,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_GetLogConfPriority(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long                   ulType,
    [in]        unsigned long                   ulTag,
    [out]        unsigned long *                  pPriority,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_AddResDes(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long                   ulLogConfTag,
    [in]        unsigned long                   ulLogConfType,
    [in]         RESOURCEID              ResourceID,
    [out]        unsigned long *                  pulResourceTag,
    [in,size_is(ResourceLen)] byte far *      ResourceData,
    [in]         PNP_RPC_BUFFER_SIZE     ResourceLen,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_FreeResDes(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long                  ulLogConfTag,
    [in]        unsigned long                  ulLogConfType,
    [in]         RESOURCEID                    ResourceID,
    [in]        unsigned long                  ulResourceTag,
    [out]        unsigned long *               pulPreviousResType,
    [out]        unsigned long *               pulPreviousResTag,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_GetNextResDes(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long                   ulLogConfTag,
    [in]        unsigned long                   ulLogConfType,
    [in]         RESOURCEID                    ResourceID,
    [in]        unsigned long                   ulResourceTag,
    [out]        unsigned long *                pulNextResDesTag,
    [out]        unsigned long *                pulNextResDesType,
    [in]        unsigned long                   ulFlags
    );

DWORD PNP_GetResDesData(
    [in]         handle_t                hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
```

```
    [in]        unsigned long             ulLogConfTag,
    [in]        unsigned long             ulLogConfType,
    [in]         RESOURCEID               ResourceID,
    [in]        unsigned long             ulResourceTag,
    [out,size_is(BufferLen)] byte far *   Buffer,
    [in]         PNP_RPC_BUFFER_SIZE      BufferLen,
    [in]        unsigned long             ulFlags
    );

DWORD PNP_GetResDesDataSize(
    [in]         handle_t                 hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long             ulLogConfTag,
    [in]        unsigned long             ulLogConfType,
    [in]         RESOURCEID               ResourceID,
    [in]        unsigned long             ulResourceTag,
    [out]        unsigned long *          pulSize,
    [in]        unsigned long             ulFlags
    );

DWORD PNP_ModifyResDes(
    [in]         handle_t                 hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]        unsigned long             ulLogConfTag,
    [in]        unsigned long             ulLogConfType,
    [in]         RESOURCEID               CurrentResourceID,
    [in]         RESOURCEID               NewResourceID,
    [in]        unsigned long             ulResourceTag,
    [in,size_is(ResourceLen)] byte far *  ResourceData,
    [in]         PNP_RPC_BUFFER_SIZE      ResourceLen,
    [in]        unsigned long                    ulFlags
    );

DWORD PNP_DetectResourceConflict(
    [in]         handle_t                 hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]         RESOURCEID               ResourceID,
    [in,size_is(ResourceLen)] byte far *  ResourceData,
    [in]         PNP_RPC_BUFFER_SIZE      ResourceLen,
    [out]        int near *               pbConflictDetected,
    [in]        unsigned long             ulFlags
    );

DWORD PNP_QueryResConfList(
    [in]         handle_t                 hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in]         RESOURCEID               ResourceID,
    [in,size_is(ResourceLen)] byte far *  ResourceData,
    [in]         PNP_RPC_BUFFER_SIZE      ResourceLen,
    [out,size_is(BufferLen)] byte far *   Buffer,
    [in]         PNP_RPC_BUFFER_SIZE      BufferLen,
    [in]        unsigned long             ulFlags
    );
```

```
void Opnum55NotUsedOnWire(void);

void Opnum56NotUsedOnWire(void);

void Opnum57NotUsedOnWire(void);

void Opnum58NotUsedOnWire(void);

void Opnum59NotUsedOnWire(void);

void Opnum60NotUsedOnWire(void);

DWORD PNP_GetCustomDevProp(
    [in]         handle_t                 hBinding,
    [in,string,ref,range(0,PNP_MAX_DEVICE_ID_LEN)] wchar_t *
        pDeviceID,
    [in,string,ref,range(0,PNP_MAX_STRING_LEN)] wchar_t *
        CustomPropName,
    [out]        unsigned long *          pulRegDataType,
    [out,size_is(*pulLength),length_is(*pulTransferLen)] byte far *
        Buffer,
    [out]        PNP_RPC_STRING_LEN      * pulTransferLen,
    [in,out]     PNP_RPC_STRING_LEN      * pulLength,
    [in]        unsigned long            ulFlags
    );

DWORD PNP_GetVersionInternal(
    [in]         handle_t                 hBinding,
    [in,out]     unsigned short           * pwVersion
    );

DWORD PNP_GetBlockedDriverInfo(
    [in]         handle_t                 hBinding,
    [out,size_is(*pulLength),length_is(*pulTransferLen)] byte far *
        Buffer,
    [out]        PNP_RPC_BUFFER_SIZE     * pulTransferLen,
    [in,out]     PNP_RPC_BUFFER_SIZE     * pulLength,
    [in]        unsigned long            ulFlags
    );

DWORD PNP_GetServerSideDeviceInstallFlags(
    [in]         handle_t                 hBinding,
    [out]        unsigned long *          pulSSDIFlags,
    [in]        unsigned long             ulFlags
    );

DWORD PNP_GetObjectPropKeys(
    [in]         handle_t                 hBinding,
    [in,string,ref,range(0,PNP_MAX_STRING_LEN)] wchar_t *
        ObjectName,
    [in]        unsigned long             ObjectType,
    [in,string,unique,range(0,PNP_MAX_CULTURE_NAME_LEN)] wchar_t *
        PropertyCultureName,
    [in,out]     PNP_PROP_COUNT          * PropertyCount,
    [out]        PNP_PROP_COUNT          * TransferLen,
    [out,size_is(*PropertyCount),length_is(*TransferLen)]
        DEVPROPKEY *PropertyKeys,
    [in]        unsigned long            Flags
```

```
    );

DWORD PNP_GetObjectProp(
    [in]         handle_t                 hBinding,
    [in,string,ref,range(0,PNP_MAX_STRING_LEN)] wchar_t *
        ObjectName,
    [in]        unsigned long             ObjectType,
    [in,string,unique,range(0,PNP_MAX_CULTURE_NAME_LEN)] wchar_t *
        PropertyCultureName,
    [in]   const DEVPROPKEY              * PropertyKey,
    [out]        DEVPROPTYPE             * PropertyType,
    [in,out]     PNP_PROP_SIZE           * PropertySize,
    [out]        PNP_PROP_SIZE           * TransferLen,
    [out,size_is(*PropertySize),length_is(*TransferLen)] byte near *
         PropertyBuffer,
    [in]        unsigned long            Flags
    );

DWORD PNP_SetObjectProp(
    [in]         handle_t                 hBinding,
    [in,string,ref,range(0,PNP_MAX_STRING_LEN)] wchar_t *
        ObjectName,
    [in]        unsigned long             ObjectType,
    [in,string,unique,range(0,PNP_MAX_CULTURE_NAME_LEN)] wchar_t *
        PropertyCultureName,
    [in]   const DEVPROPKEY               * PropertyKey,
    [in]         DEVPROPTYPE              PropertyType,
    [in]         PNP_PROP_SIZE            PropertySize,
    [in,unique,size_is(PropertySize)] byte near * PropertyBuffer,
    [in]        unsigned long             Flags
    );

}
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows NT
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1: Version 0.0 of the Plug and Play Remote (PNPR) Protocol is implemented by Windows NT 4.0 only. Version 1.0 is implemented by Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<2> Section 1.3: Windows implementation uses the RPC protocol to retrieve the identity of the caller, as specified in [MS-RPCE] section 3.3.3.4.3. The server uses the underlying Windows security subsystem to determine the permissions for the caller. If the caller does not have the required permissions to execute a specific method, the method call fails and returns CR_ACCESS_DENIED (0x00000033).

The specific security check implemented by the server to determine if the client has access to call any given interface method varies among each of the methods and among different implementations of the protocol. Generally, the methods are determined to either require special privileges, special group access rights, or both. Below are the minimum security requirements implemented for all methods of the interface on each version of Windows:

- Windows NT, Windows 2000, Windows XP, and Windows XP SP1: The client MUST authenticate to the server as a member of the local **Authenticated Users** group to call any method of the PnP interface.
- Windows XP SP2, Windows Server 2003, and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group to call any method of the PnP interface.

Additional differences in the security requirements for specific interface methods are defined elsewhere throughout this document.

<3> Section 1.9: \\PIPE\plugplay is used on the Windows Vista version of the server, for all opnums. \\PIPE\ntsvcs may also receive requests for methods up to and including opnum 64 (**PNP_GetServerSideDeviceInstallFlags**). Methods with opnum 65 (**PNP_GetObjectPropKeys**) and greater are only accessible through the \\PIPE\plugplay endpoint.

<4> Section 2.1: The \\PIPE\plugplay well-known endpoint is used by the Windows Vista and Windows Server 2008 implementations of the server, for all opnums. \\PIPE\ntsvcs may also receive requests for methods up to and including opnum 64 (**PNP_GetServerSideDeviceInstallFlags**). Methods with opnum 65 (**PNP_GetObjectPropKeys**) and greater are only accessible through the \\PIPE\plugplay endpoint.

<5> Section 2.2.9: Windows NT, Windows 2000, Windows XP, Windows Server 2003: Interface element size limits are not implemented by the protocol for these versions. No interface method parameters are decorated with the RPC [range] attribute.

<6> Section 3.1.3: Windows Vista: The Plug and Play Remote (PNPR) Protocol is not available in default configurations. It can be enabled through the Windows Group Policy infrastructure.

<7> Section 3.1.4: Windows 2000, Windows XP, and Windows Server 2003: This protocol MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 5.0, as specified in [MS-RPCE] section 3.

Windows NT: This protocol disables strict checking to enforce NDR data consistency. The RPC runtime does not perform a strict data consistency check, as specified in [MS-RPCE] section 3. This protocol MUST indicate to the RPC runtime via the strict_context_handle attribute that it is to reject use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

<8> Section 3.1.4: Windows NT, Windows 2000, Windows XP, and Windows XP SP1: The client MUST authenticate to the server as a member of the local authenticated users group to call any method of the PNPR Protocol interface.

Windows XP SP2 and later, Windows Server 2003, and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group to call any method of the PNPR Protocol interface.

<9> Section 3.1.4: Gaps in the opnum numbering sequence apply to Windows as follows.

Opnum	Description
00	Only used locally by Windows, never remotely.
01	Only used locally by Windows, never remotely.
04	Only used locally by Windows, never remotely.
05	Only used locally by Windows, never remotely.
55	Just returns ERROR_NOT_IMPLEMENTED. It is never used.
56	Just returns ERROR_NOT_IMPLEMENTED. It is never used.
57	Just returns ERROR_NOT_IMPLEMENTED. It is never used.
58	Just returns ERROR_NOT_IMPLEMENTED. It is never used.
59	Only used locally by Windows, never remotely.
60	Only used locally by Windows, never remotely.

<10> Section 3.1.4.1: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<11> Section 3.1.4.2: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<12> Section 3.1.4.3: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<13> Section 3.1.4.4: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<14> Section 3.1.4.4: The root of the hardware tree is HTREE\ROOT\0.

<15> Section 3.1.4.5: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<16> Section 3.1.4.6: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<17> Section 3.1.4.9: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<18> Section 3.1.4.10: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<19> Section 3.1.4.11: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

<20> Section 3.1.4.12: This string represents a registry subkey under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class.

<21> Section 3.1.4.13: Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

Windows uses the registry for device-specific storage locations.

<22> Section 3.1.4.14: Windows NT, Windows 2000, and Windows XP: The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

Windows uses the registry for device-specific storage locations.

<23> Section 3.1.4.15: This method requires Windows NTWindows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows Vista: This method is not implemented, and returns CR_CALL_NOT_IMPLEMENTED.

<24> Section 3.1.4.16: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<25> Section 3.1.4.17: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows NT, Windows 2000, and Windows XP: The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

Windows uses the registry for device-specific storage locations.

<26> Section 3.1.4.17: Windows Vista and Windows Server 2008: Storage locations for devices can only be deleted by specifying this flag if all such devices have a status of DN_WILL_BE_REMOVED.

<27> Section 3.1.4.18: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<28> Section 3.1.4.19: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<29> Section 3.1.4.20: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<30> Section 3.1.4.21: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows NT, Windows 2000, and Windows XP: The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

<31> Section 3.1.4.22: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows NT, Windows 2000, and Windows XP: The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

<32> Section 3.1.4.23: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<33> Section 3.1.4.24: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows NT, Windows 2000, and Windows XP: The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

<34> Section 3.1.4.25: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows NT, Windows 2000, and Windows XP: The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

<35> Section 3.1.4.25: The parent of the new device MUST be HTREE\ROOT\0.

<36> Section 3.1.4.26: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

<37> Section 3.1.4.27: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<38> Section 3.1.4.28: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

<39> Section 3.1.4.29: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

<40> Section 3.1.4.30: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows NT, Windows 2000, and Windows XP: The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

<41> Section 3.1.4.31: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows NT, Windows 2000, and Windows XP: The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

<42> Section 3.1.4.32: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

<43> Section 3.1.4.33: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

<44> Section 3.1.4.34: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

When called for a device that does not represent a docking station, the client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege. When called for a device that is a dock station, the client MUST authenticate to the server as a user who is granted the SeUndockPrivilege privilege.

<45> Section 3.1.4.35: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<46> Section 3.1.4.36: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

The client MUST authenticate to the server as a user who is granted the SeUndockPrivilege privilege.

<47> Section 3.1.4.37: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

When called with *ulAction* set to PNP_SET_HWPROFFLAGS, the client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

<48> Section 3.1.4.38: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<49> Section 3.1.4.39: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows NT, Windows 2000, and Windows XP: The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

<50> Section 3.1.4.40: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows NT, Windows 2000, and Windows XP: The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

<51> Section 3.1.4.41: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<52> Section 3.1.4.42: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<53> Section 3.1.4.43: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<54> Section 3.1.4.44: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows NT, Windows 2000, and Windows XP: The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

<55> Section 3.1.4.45: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows NT, Windows 2000, and Windows XP: The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

<56> Section 3.1.4.46: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<57> Section 3.1.4.47: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<58> Section 3.1.4.48: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<59> Section 3.1.4.49: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows NT, Windows 2000, and Windows XP: The client MUST authenticate to the server as a user who is granted the SeLoadDriverPrivilege privilege.

Windows Server 2003 and Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

<60> Section 3.1.4.50: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

Windows XP and Windows Vista and Windows Server 2008: Returns CR_CALL_NOT_IMPLEMENTED (0x00000034). The call is not implemented.

<61> Section 3.1.4.51: This method requires Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0 SP3 and later, Windows 2000 Server, Windows Server 2003, or Windows NT Server 4.0 SP3 and later.

<62> Section 3.1.4.52: This method only available for Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<63> Section 3.1.4.52: The *CustomPropName* string is the name of a registry value that can be set in the Device Parameters subkey for device settings.

<64> Section 3.1.4.53: This method only available for Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<65> Section 3.1.4.54: This method is available only for Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<66> Section 3.1.4.55: This method is available only for Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<67> Section 3.1.4.56: This method is available only for Windows Vista and Windows Server 2008.

<68> Section 3.1.4.57: This method is available only for Windows Vista and Windows Server 2008.

<69> Section 3.1.4.58: This method is available only for Windows Vista and Windows Server 2008.

Windows Vista: The client MUST authenticate to the server as a member of the local administrators group.

<70> Section 3.2.4: Windows 2000, Windows XP, and Windows Server 2003: This protocol MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 5.0, as specified in [MS-RPCE] section 3.

8 Index

N

O

P

R

S

T

V

W